Manipulation de diagrammes UML

TER L3 Informatique

Mai 2008

G. Dalichoux, G. Delmas, F. Hervouet, L. Muthelet

Table des matières

Résumé

Mots-clefs : Yaut, UML, Java, MVC

Ce projet est l'aboutissement d'un peu moins de trois mois de travail de quatre étudiants de l'Université Montpellier II, réalisé durant le second semestre de la troisième année de Licence. Il s'agit d'un logiciel appelé Yaut (Yet Another UML Tool), permettant de modéliser de manière simple et efficace des diagrammes de classes à la norme UML. Mais son ingéniosité réside plutôt dans la possibilité d'exporter et d'importer du code source dans différents langages (Java, C++, PHP5, Python). D'autant plus que l'exportation est laissée à l'appréciation des utilisateurs, puisqu'elle suit un modèle basé sur une structure XML modifiable à volonté, et qui laisse donc la possibilité aux utilisateurs de rajouter des langages supportés.

Le projet a suivi une phase de développement incluant une étude de l'existant, une analyse UML, puis une programmation orientée objet en Java, en suivant un design pattern MVC.

Abstract

Keywords : Yaut, UML, Java, MVC

This project has been developped for three months by four students of the french University of Montpellier II, during the third degree second half-year. This software called Yaut (Yet Another UML Tool), allows you in a simple, easy and effective way to design class diagrams to the UML standard. But in fact, its interest lies rather in the possibility of exporting and importing source code under various languages (Java, C++, PHP5, Python). Moreover, the exportation is adaptable to all users, because it follows XML structures, which are editable, and allows users to add other languages for being supported by the software.

The development of the project included an existing study, a UML analysis and design, and after that an oriented object programmation under the Java language, with the MVC design pattern.

Remerciements

Nous souhaitons remercier vivement quelques personnes qui nous ont permis de réaliser ce TER durant notre troisième année de Licence.

La première personne que nous remercions tout particulièrement est bien évidemment notre tuteur Michel Meynard, qui a accepté l'idée de ce projet lorsque nous la lui avons proposée. Il nous a permis de ne pas nous égarer, tout en nous permettant d'avancer et de réussir à réaliser des éléments du cahier des charges qui n'étaient destinés qu'à être développés si le planning avait de l'avance.

Les remerciements suivants se tournent vers Marianne Huchard, professeur de l'UE de Programmation objet ce semestre. Elle a accepté de répondre à nos interrogations sur la manière de conceptualiser notre logiciel, et nous a amenés à nous diriger vers une application du modèle de conception MCV (Model/View-Controller).

Enfin, nous remercions vivement Gilles Simonin, notre chargé de TD de Programmation Objet, qui lui, nous a permis de résoudre des problèmes d'ordre purement technique en programmation (pour l'exportation de code en C++ notamment).

Glossaire

IDE : "Integrated Development Environment" (ou "Environnement de Développement Intégré"). Programme regroupant un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur.

Java : Java est à la fois un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable créé *Sun Microsystems* présenté officiellement en Mai 1995. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que *Unix*, *Microsoft Windows*, *Mac OS* ou *Linux* avec peu ou pas de modifications. C'est la plate-forme qui garantit la portabilité des applications développées en Java. Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs.

Programmation orientée objet : Paradigme de programmation informatique qui consiste en la définition et l'assemblage de briques logicielles appelées objet ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre.

Reverse engineering : (ou rétroconception). Activité qui consiste à étudier un objet pour en déterminer le fonctionnement interne ou sa méthode de fabrication.

Swing : Bibliothèque graphique pour le langage de programmation Java. Swing offre la possibilité de créer des interfaces graphiques identiques quel que soit le système d'exploitation sous-jacent. Il utilise le principe Modèle/Vue-Contrôleur et dispose de plusieurs choix d'apparence pour chacun des composants standards.

UML : "Unified Modeling Language" (ou "langage de modélisation unifié"). Langage graphique de modélisation des données et des traitements. C'est une formalisation très aboutie et non-propriétaire de la modélisation objet utilisée en génie logiciel.

XML : "eXtensible Markup Language" (ou "langage de balisage extensible"). Langage informatique de balisage générique. Son objectif initial est de faciliter l'échange automatisé de contenus entre systèmes d'informations hétérogènes (interopérabilité).

Chapitre 1

Introduction

1.1 Généralités

Dans notre cursus de Licence à la faculté de Sciences de Montpellier, il est bon pour nous d'avoir un contact avec ce que peut représenter la gestion et la réalisation complète d'un projet. C'est pourquoi en L3 nous sont proposés des sujets de TER par les enseignants, visant à nous familiariser avec la conduite de projet, puisque cela nous attendra forcément dans notre vie professionnelle.

Ce TER est une Unité d'Enseignement (UE) à part entière et compte pour 6 crédits ECTS, et dure donc le temps d'un semestre. Il n'y a pas de créneau horaire de présence obligatoire à l'université concernant cette UE, ce qui permet donc aux étudiants de travailler où et quand ils le souhaitent. L'encadrement se fait par un professeur qui joue le rôle de tuteur dans le but de guider les étudiants dans leur démarche.

Lorsque la liste des sujets proposés a été disponible, aucun de nous n'avait réellement trouvé son bonheur dans celle-ci. Les quelques jours de réflexion qui ont suivi ne nous ont pas plus convaincus, mais nous ont en revanche appris que nous pouvions nous-mêmes proposer un sujet. Cette idée a mûri dans nos têtes et nous a amenés à réfléchir sur nos besoins quotidiens en tant qu'étudiants en informatique.

C'est après mûre réfléxion que nous avons réalisé qu'aucun de nous ne connaissait un outil de modélisation UML gratuit, open-source, bien pensé, et permettant d'exporter du code. Nous avons donc soumis l'idée à Michel Meynard, qui nous a donné son aval. Nous avons donc enclenché la phase d'analyse de l'existant et la rédaction d'un cahier des charges pour savoir à quoi nous en tenir.

1.2 Le sujet

C'est donc tout naturellement sur cette idée que nous sommes partis : un logiciel réalisé par et pour des étudiants en informatique, dans le but de faciliter un travail qui est le point de départ de toute application naissante.

UML est une méthode d'analyse très performante et surtout très utilisée, entre autres, dans la programmation orientée objet. Offrir la possibilité de créer, éditer et manipuler des diagrammes comme les diagrammes de classes serait donc une bonne idée.

Ayant nous-mêmes proposé le sujet, nous avions posé comme bases des fonctionnalités relativement courantes pour ce type de logiciel, axé particulièrement sur la représentation graphique des

diagrammes, et sur la génération de code relative aux objets ainsi créés.

Le sujet était particulièrement vaste et surtout très flou dans nos têtes, c'est pourquoi nous l'avons proposé volontairement de manière très ouverte, afin de pouvoir nous l'approprier une fois les premières étapes d'analyse effectuées. C'était notre manière à nous, du fait que nous ne nous rendions pas réellement compte de la quantité de travail par rapport au temps imparti, de nous laisser le choix plus tard d'adapter le sujet selon les points qui nous semblaient réalisables mais également enrichissants des points de vue programmation.

1.3 Cahier des charges

1.3.1 Analyse de l'existant

Effectuer une analyse des outils UML déjà existants était un point essentiel dans le bon déroulement de ce projet. Nous avons chacun recherché et testé des logiciels qui nous ont permis d'avancer dans la visualisation globale de notre projet.

Les produits testés étaient, en proportions à peu près équivalentes, pour certains sous licence GNU/GPL, et d'autres totalement propriétaires et payants (chers qui plus est).

Notre analyse a révélé que les logiciels payants étaient très souvent vraiment complets, mais au détriment de la légèreté, de la facilité d'emploi, et de la prise en main. Des logiciels comme *Poseidon*, *Rational Rose*, ou encore *Power AMC*, sont autant de références dans l'entreprise, qui ne le sont pas entre les mains d'étudiants. Les fonctionnalités exhaustives font perdre en clarté pour un utilisateur occasionnel.

Les logiciels libres de notre analyse n'étaient cependant pas tous attirants pour autant. En effet, certains logiciels libres étaient assez peu pratiques d'utilisation, ou alors n'étaient esthétiquement pas enviables (*ArgoUML*), mais il faut reconnaître que la grande majorité de ceux testés avaient été conçus de manière portable, à savoir pour plusieurs systèmes d'exploitation, ce qui est toujours très agréable. Il est à noter que nous avons trouvé un logiciel sous licence libre, très complet, très efficace, nommé *Umbrello*, à qui, il faut bien le reconnaître, nous n'avons pas trouvé de défaut majeur, malgré quelques lacunes dans l'exportation de code.

1.3.2 Fonctionnalités

L'analyse des applications permettant la manipulation de diagrammes UML nous a conduits à revoir le sujet de base pour mieux nous l'approprier, étant donné le temps qui nous était imparti ainsi que les fonctionnalités que nous souhaitions réellement mettre en place dans ce projet. Nous avons donc borné le sujet originel, mais avec des balises qui nous étaient propres.

L'idée que nous nous sommes faîte par rapport au sujet de départ, est qu'il nous fallait nous tourner vers la création d'une solution destinée aux étudiants qui souhaiteraient allier une bonne modélisation par un diagramme des classes, à une exportation de code, dans le but de faciliter la programmation et de gagner du temps. Nous avons donc effectivement abandonné l'idée qu'il nous fallait donner la possibilité de créer plusieurs types de diagrammes, car des logiciels payants étaient tellement complets, qu'il nous était impossible en trois mois de projet de rivaliser.

Nous avons également décidé de tenter l'expérience du "reverse engineering", ou rétro-conception, qui pourrait permettre d'importer un projet complet (sous forme de code source) dans notre logiciel, afin d'en visualiser les diagrammes de classes associés.

Obligatoires

Les fonctionnalités obligatoires seront donc celles que nous décidons de mettre en place coûte que coûte :

- Création/Sauvegarde/Chargement de diagrammes des classes sous forme graphique et assistée
- Interface ergonomique permettant une création et une manipulation aisée des éléments composant un diagramme des classes
- Interface claire et soignée permettant à un utilisateur de ne pas être perdu lors du premier contact
- Exportation du diagramme des classes sous forme d'image (JPG)
- Exportation sous forme de code du diagramme (Java, C++)
- Importation de code pour la création d'un diagramme des classes (Java, C++)
- Possibilité pour les utilisateurs d'augmenter facilement (sous forme de fichier XML structurés) le nombre de langage supportés pour l'exportation du code

Optionnelles

Les fonctionnalités optionnelles sont quant à elles celles que nous programmerons si il nous reste du temps, et si les tests des fonctionnalités de base sont concluants :

- Support de plus de langages pour l'exportation
- Support de plus de langages pour l'importation
- Support de plus de formats pour l'exportation sous forme d'image

Il nous faudra bien évidemment faire en sorte que les fonctionnalités obligatoires marchent parfaitement bien, avant de mettre en place les optionnelles, plutôt que d'avoir un produit bancal qui paraîtrait non fini.

Chapitre 2

Organisation

Quelle a été l'organisation de notre travail ? Comment s'est effectuée la distribution des tâches ? Quelles tâches nous ont pris le plus de temps ? Quels outils avons-nous utilisés ?

2.1 Organisation du travail

2.1.1 Humaine

Une décomposition équitable s'est opérée rapidement et d'elle-même, par rapport au découpage du projet.

En effet, nous avons souhaité découper le projet en deux parties distinctes que sont le noyau de l'application, et la partie graphique, dans un souci de clarté et d'efficacité pour une possible réutilisation. Cette idée de base pour une saine construction du projet nous a naturellement amenés à scinder notre groupe en deux : Laurent & Fabien pour le noyau de l'application, et Guillaume & Gabriel pour l'implémentation graphique de l'application.

Le choix d'un chef de projet s'est également imposé de lui-même. Ce sera donc Fabien qui s'occupera de veiller à ce que tout le monde effectue les tâches qui lui sont dévolues et respecte au maximum les délais.

Nous savons malgré cette organisation interne, que nous devons tous avoir une vision globale du projet que nous mettons en place, car même si nous n'apportons qu'une pierre à l'édifice, nous savons par expérience, que cette pierre sera mieux placée si elle s'inscrit dans une démarche globale.

C'est la raison pour laquelle nous effectuerons un travail d'analyse en respectant ces groupes, mais que nous mettrons tout cela en commun pour recueillir les avis et suggestions de chacun.

2.1.2 Temporelle

L'organisation temporelle du travail est toujours relativement difficile à prévoir, mais il nous faut tout de même essayer de nous fixer des dates limites, ce qui nécessite une analyse en aval des tâches à effectuer.

Nous avons souhaité découper notre planification des tâches en trois parties, qui seront toutes trois modifiées au fur et à mesure de l'avancée, et ce jusqu'à la fin du projet. Ces trois parties sont lies à l'idée même du développement d'un projet informatique : l'analyse, la programmation, ainsi que la rédaction des documents liés.

+ **Analyse** :
  – Analyse de l'existant (1 semaine)

  - Analyse UML de la partie noyau (2 semaines)
  - Analyse UML de la partie graphique (2 semaines)
  - Définition d'une IHM (1 semaine)
- + **Programmation** :
  - Programmation de la version 1.0 de la partie noyau (5 semaines)
  - Programmation de la partie 1.0 graphique (6 semaines)
  - Tests, traque des bugs et autres oublis (2 semaines)
  - Finalisation de l'application en vue de la date limite de livraison (1 semaine)
- + **Rédaction des documents liés** :
  - Rédaction du cahier des charges (1 semaine)
  - Rédaction du rapport du projet (11 semaines)
  - Finalisation du dossier d'analyse (1 semaine)
  - Rédaction de l'aide et du manuel d'utilisation du logiciel (1 semaine)

2.2 Choix des outils de développement

Dans cette partie, nous allons expliciter les choix que nous avons été amenés à faire dans la sélection des outils propres au développement du projet, en allant du langage de programmation à l'outil de rédaction du rapport.

2.2.1 Java & Eclipse

Une des raisons majeure qui nous a poussés à développer Yaut en Java, c'est que celui-ci est d'une portabilité à toute épreuve pour peu que la machine virtuelle Java (JVM) soit installée sur le système d'exploitation (OS) de l'utilisateur, et étant donné que cette JVM est proposée sur les trois OS principaux - à savoir Windows, Linux, MacOS - cela augmente considérablement la cible potentielle d'utilisateurs.

Après viennent les choix purement en rapport avec la façon de programmer. Nous souhaitions bien évidemment développer en utilisant la programmation orientée objet, paradigme fondateur de ce langage, puisqu'en Java, tout est objet, et nous connaissions tous les quatre ce langage rencontré au cours de notre cursus.

Le choix de l'environnement de développement (IDE) s'est fait de lui-même, nous étions trois sur quatre à connaître et avoir utilisé Eclipse à plusieurs reprises. Celui-ci est réellement efficace pour la gestion de projets complexes. Il a également le mérite de proposer des fonctions que d'autres éditeurs ne font pas toujours, ou pas bien, comme l'auto-complétion, qui fait gagner un temps précieux aux développeurs. De plus il est configurable de fond en comble et permet même l'ajout de nouvelles fonctionnalités par un système de plugins, chose particulièrement agréable.

2.2.2 Subversion

Il arrive un moment dans la réalisation d'un projet à plusieurs où le problème du développement à distance se pose. En effet, lorsque plusieurs personnes sont amenées à travailler sur les mêmes fichiers, il faut obligatoirement centraliser les développements effectués par chacun, mais en prenant bien soin de ne pas écraser les modifications faites par l'un ou par l'autre.

Et c'est précisément le rôle de Subversion (SVN), un système de gestion de versions concurrentes côté serveur. Il permet donc aux développeurs de récupérer toute l'arborescence du code source présente sur le dépôt, mais également de mettre en ligne si des changements ont été appliqués aux

fichiers. Il prévient les écrasements en informant les utilisateurs et en leur demandant de régler les conflits, et garde un historique complet des changements effectués depuis la création du projet.

Au commencement du projet, nous cherchions des sites proposant gratuitement un serveur de gestion de projet type SVN, et nous en avions trouvé plusieurs. Nous avons retenu GoogleCode, qui avait toute notre confiance du point de vue de la fiabilité.

2.2.3 JFlex

Etant donné que nous avions besoin de faire de la reconnaissance d'expressions dans des fichiers texte pour pouvoir réaliser du reverse engineering, nous ne devions pas négliger cet aspect. Il se trouve que l'outil *Flex*, utilisant les expressions régulières, que nous avions utilisé en TP d'Interprétation & Compilation pour le langage C, a été porté pour Java. Les conventions d'écriture des sources *Flex* étant les mêmes que celles que nous avions vues en TP, l'adaption fut aisée.

Il existait par ailleurs un équivalent de Yacc/Bison, appelé Cup, qui permettait donc de gérer la reconnaissance par des grammaires, mais nous avons préféré écarter ce choix, qui nous semblait moins adapté à la situation.

2.2.4 JDOM

Pour la manipulation de fichiers à la norme XML nous avons tout simplement utilisé JDOM. C'est une bibliothèque open-source permettant de parser des fichiers XML, créée spécifiquement pour la plateforme Java. Il utilise des collections SAX (Simple API for XML) pour l'analyse. Son intérêt réside dans le fait que ses créateurs ont réussi le pari de rendre simple et efficace le fait de développer une application complexe autour de XML avec DOM.

2.2.5 LaTeX

Pour la rédaction de ce rapport, nous avons préféré l'utilisation de LaTeXà celle d'un WYSIWYG classique tel que *Microsoft Word* ou autre *OpenOffice.org*. Ce choix a été motivé par l'idée que nous devions avant tout nous occuper du contenu et non pas de sa mise en page. En effet, LaTeXs'occupe seul de la mise en page, et produit des documents standards qui sont étudiés pour être très lisibles, tout en respectant une certaine sobriété.

De plus il est fort probable que nous ayons de nouveau à produire documents dans ce format, ce qui nous paraîtra alors plus facile et efficace.

Chapitre 3

Analyse

L'analyse est la phase qui suit l'idée de projet. En effet, elle est extrêmement importante et nécessite un travail abstrait qui se révèle être nécessaire pour une bonne réalisation. Nous avons limité notre analyse à deux types de diagramme UML, à savoir le diagramme des cas d'utilisation, et le diagramme des classes.

3.1 Diagramme des cas d'utilisation

Nous avons donc tout d'abord raisonné du point de vue de l'utilisateur. En effet, il est impératif lors de la création d'un logiciel, étant donné qu'il est associé à un besoin de la part d'un futur utilisateur, de se placer de son point de vue, et de réfléchir aux possibilités qu'il nous faut donner à ce nouveau programme.

Etant donné le fait que nous étions nos propres demandeurs, et que nous serions nos premiers utilisateurs, cela facilitait grandement le travail. C'est d'ailleurs cette même raison qui nous a poussés à proposer un maximum de fonctionnalités, et surtout à rassembler les meilleures parmi celles proposeés par les autres logiciels croisés au hasard de travaux pratiques.

Comme le diagramme parle de lui-même, il n'est pas nécessaire de s'y attarder (Cf. aux annexes).

3.2 Diagrammes des classes

Nous allons expliciter ici les différents choix de hiérarchisation objet que nous avons faits tant au niveau du noyau qu'à celui de la partie graphique.

3.2.1 Noyau

Le noyau représente la base de l'application, ce qui lui permet d'exister. Il est l'architecture fondatrice d'un diagramme des classes UML. Puisque nous souhaitions en modéliser, il nous a fallu effectuer une mise à plat de tous les objets pouvant graviter autour de leur création.

Nous avons très vite réussi à réaliser un squelette de modélisation, qui a par la suite été complété, mais jamais remis en cause.

+ **DiagrammesClasses** : Cette classe est l'objet principal de ce diagramme. Elle modélise un diagramme des classes avec tout ce qui le compose dans la norme UML, à savoir une liste de classes, une liste de liens d'héritages, ainsi qu'une liste de liens d'agrégations.

+ **Classe** : Une *Classe* a, outre les banalités d'usage telles qu'un nom et une description, une visibilité (publique, privée ou protégée), peut être abstraite ou non, et comporte deux listes : une d'attributs, et une de méthodes.
+ **Variable** : Une *Variable* modélise ce qu'il y a de plus simple pour une variable au sens programmation du terme, elle met donc en jeu un type et un nom.
+ **Attribut** : *Attribut* est une classe qui spécialise Variable. En effet, un *Attribut* a toutes les qualités d'une variable, mais possède également une visibilité dans la classe (publique, privée ou protegée), ainsi qu'une description.
+ **Methode** : Une *Méthode* peut être abstraite, a une visibilité, un nom, la description de son action. Elle a également type de retour, ainsi qu'une liste de paramètres (représentés par l'objet *Variable*).
+ **Lien** : Un diagramme des classes ne serait rien sans liens. Cette classe permet de modéliser un lien entre deux classes, et va donc nous permettre de modéliser différents liens, que ce soit un héritage ou une agrégation.

3.2.2 Graphique

Chacun des objets d'un diagramme des classes va devoir être représenté graphiquement sur la scène.

Java permet de définir des interfaces, une classe totalement abstraite, dans laquelle aucune méthode n'est définie. On définit alors une interface IDessinableManipulable qui devra être implémentée par tous les objets graphique contenus sur la scène.

Nous allons définir deux types d'éléments différents : les composants, et les liens. Ce choix est directement tiré de la modélisation du noyau, et nous semblait tout à fait logique.

+ Composants : Ils représenteront les classes et les notes sur la scène.
+ Liens : Pour les héritages et les agrégations.

Nous avons donc modélisé deux classes pour suivre ce schéma de conception : *RepresentationComposant* et *RepresentationLien* qui implémenteront bien évidemment l'interface *IDessinableManipulable.*

La classe SceneEditeur possède donc une liste d'objets du type *IDessinableManipulable*, qui vont pouvoir être dessinés, et qui donneront des possiblités d'interaction à l'utilisateur.

3.2.3 Utilisation du patron MVC

> "*Le Modèle-Vue-Contrôleur (MVC) est une architecture et une méthode de conception qui organise l'interface Homme-machine d'une application logicielle. Il divise l'ihm en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des évènements, synchronisation), chacun ayant un rôle précis dans l'interface.*" Encyclopédie Wikipédia.

Cette définition résume parfaitement l'intérêt de ce patron de conception. En effet, le fait de penser une application en séparant complètement le fond de la forme, permet d'une part de ne pas altérer le fond qui reste donc intègre, mais d'autre part de proposer plusieurs formes par la suite.

La séparation du modèle (à savoir les différentes classes nous permettant de simuler un diagramme des classes UML), et de sa représentation (la scène permettant la création et la manipulation des diagrammes), permettra une meilleure réutilisation du code si la représentation souhaitée ne convient plus.

C'est précisément pour cette raison que nous avons choisi (en suivant de bons conseils) d'appliquer ce patron de conception à notre application.

Chapitre 4

Développement

Nous avons souhaité développer plus amplement, dans cette partie, nos choix de programmation, la façon dont nous avons envisagé les choses, les problèmes auxquels nous avons été confrontés, et comment nous les avons résolus.

4.1 L'interface graphique

L'interface générale d'un logiciel représente une partie délicate de la réalisation. Il faut qu'elle soit intuitive, claire, tout en respectant une certaine sobriété. La modélisation de cette interface ne nous a pas pris beaucoup de temps, puisque les principaux éléments nécessaires nous sont apparus rapidement : des menus pour accéder aux différentes commandes, une scène servant à manipuler les différents objets d'un diagramme, un arbre contenant tous les objets de la session (classes, héritages, agrégations), ainsi qu'une fenêtre retraçant toutes les actions effectuées par l'utilisateur. Ces éléments sont tous développés plus bas.

Nous souhaitions par ailleurs permettre à l'utilisateur de réaliser plusieurs diagrammes sans avoir à lancer plusieurs fenêtres de l'application. C'est donc tout naturellement que nous nous sommes tournés vers la mise en place d'onglets. Cela permet de gérer des projets complets d'analyse, en conservant de la clarté. Et c'est donc dans cette optique que nous avons décidé d'insérer la notion de projet dans notre application. Un projet est en fait un ensemble de diagrammes réalisés dans des onglets différents, et peut être sauvegardé et/ou importé.

4.1.1 La scène

Réalisation

Nous avons décidé de spécialiser la classe *JPanel* pour modéliser la scène (l'avantage majeur qui nous a fait utiliser des composants standards de la bibliothèque Swing est la prise en charge intégrée du double buffering). Ainsi pour dessiner, il nous suffira de surcharger la méthode *paintComponent(Graphics g).*

Tous les objets qui gravitent sur la scène se doivent avoir les mêmes propriétés :
+ Sélection de l'objet
+ Récupération/Modification de ses coordonnées
+ Recherche d'intersection par rapport à certaines coordonnées

Les objets sélectionnés disposent de 4 points (situés aux extrémités) permettant de représenter visuellement la sélection. Ces 4 points sont aussi présents pour permettre le redimensionnement du composant.

De plus ils doivent chacun avoir un menu contextuel personnalisé au clic droit, ainsi qu'éventuellement des actions prédéfinies au double-clic.

Pour permettre la manipulation des objets de la scène, il nous a été nécessaire d'utiliser les écouteurs souris (MouseMotionListener, MouseAdapter). De ce fait, le scénario du déplacement d'une figure devient celui-ci : l'utilisateur clique sur la scène, les coordonnées du point cliqué sont transmises à toutes les figures, et c'est la première qui répond au fait que le point est contenu dans sa zone de dessin qui passe au stade "sélectionnée". Si l'utilisateur conserve le clic et qu'il déplace la souris, la figure se déplace en suivant les mouvements de la souris, et ce jusqu'au relâchement par l'utilisateur.

Les liens mettent en jeu deux composants, et leurs points d'ancrage se situent par rapport au centre de ces composants.

Il est d'ailleurs intéressant de noter que nous avons dû créer deux classes concernant les liens : Triangle et Losange. Elles ont été destinées uniquement à l'affichage respectif d'un triangle et d'un losange pour la représentation symbolique de l'héritage et de l'agrégation. Néanmoins pour les afficher correctement, il nous a fallu user de calculs de rotations. Voici classées, les différentes fonctionnalités que nous avons décidé d'ajouter à la scène, dans le but de rendre l'utilisation plus agréable pour l'utilisateur.

Fonctionnalités

Voici classées, les différentes fonctionnalités que nous avons choisi de mettre en place pour la scène.

Exportation image : Dès lors qu'un diagramme des classes est réalisé, c'est non seulement dans une perspective de préparation de programmation, mais également dans le but de réaliser un dossier d'analyse, qui sera la plupart du temps adjoint au rapport concernant l'application. C'est ce constat qui nous a amenés à souhaiter pouvoir sauvegarder en image la scène. Le format choisi a été le JPG, de par le fait qu'il soit très répandu, et surtout car Java gère ce type d'image en natif.

Dans la pratique, la classe SceneEditeur possède une méthode *getImage()* qui renvoie une BufferedImage. Celle-ci est construite par rapport au positionnement des composants.Nous avons cherché à rcupérer les coordonnées les plus éloignées aux quatre angles du diagramme, dans le but d'obtenir un cadre contenant tous les composants. Puis il nous a suffi ensuite de rajouter une marge pour obtenir l'image souhaitée.

Utilisation du clavier : L'utilisation du clavier permet de mettre en place des raccourcis pour réaliser certaines actions répétitives, ou du moins qui sont communément utilisées dans l'informatique. Nous avons décidé de permettre à l'utilisateur d'utiliser, en plus du déplacement à l'aide de la souris, un déplacement des composants sur la scène à l'aide des touches directionnelles, pour une des raisons de précision, car cela permet un déplacement au pixel près.

C'est grâce à l'interface *KeyListener*, qui permet à une classe de répondre aux évènements liés au clavier (appui sur une touche, relâchement, etc...), que nous avons réussi ceci.

Menus contextuels : Chaque élément de la scène possède un menu contextuel accessible à partir du clic droit. Il permet de réaliser directement des actions sur le composant sélectionné, comme changer sa couleur de fond, modifier son contenu ou encore le supprimer.

4.1.2 L'arbre de visualisation

Fonctionnement

L'arbre permet de visualiser sous forme d'arborescence toutes les classes qui ont été créées ou importées, ainsi que les héritages et les agrégations.

L'avantage de ce choix de conception était de laisser la possibilité de visualiser tous les objets en un seul coup d'oeil, puisque même si l'utilisateur travaille sur plusieurs diagrammes, tous les objets sont regroupés au même endroit. D'ailleurs l'autre possibilité intéressante que nous souhaitions mettre en place, était de permettre la réutilisation des objets, rendue possible puisque les objets de l'arbre peuvent être utilisés dans tous les diagrammes en cours.

En pratique, l'arbre est une classe spécialisant JTree, qui permet, comme nous le désirions, de visualiser une structure d'arbre. Les références des différents objets sont passées à notre arbre pour qu'il puisse en afficher les caractéristiques. Dès qu'une classe est modifiée, l'objet va être recherché dans l'arbre, en utilisant la méthode *equals(Object obj)*, puis mis à jour.

Là encore, nous avons donné la possibilité à l'utilisateur d'interagir avec les objets contenus via un menu contextuel au clic droit, permettant notamment la suppression.

Personnalisation de l'arbre

Dans le but de rendre plus agréable et plus claire la lisibilité de l'arbre, nous avons mis en place un système d'icône. En effet, pour identifier les classes, leurs attributs et leurs méthodes, chacun de ces trois types d'objets a une icône qui lui est dédiée, et qui prend en compte sa visibilité (privé, public, protégé). La mise en place de ces icônes est rendue possible par l'utilisation de la classe *DefaultTreeCellRenderer*, en surchargeant la méthode *getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)*. Après il suffit d'afficher l'icône correspondant au type de l'instance de l'élément contenu dans l'arbre.

4.1.3 Autres

Le contrôleur

Le contrôleur est l'élément essentiel qui permet d'appliquer le patron MVC que nous avions décidé d'utiliser. Il est le maillon qui va permettre de maintenir de la cohérence entre le modèle, la scène, et l'arbre.

Etant donné que ces trois éléments lui sont passés en paramètres du constructeur, il les connaît et sera donc appelé à chaque demande de modification sur l'une des deux vues. Et c'est à ce moment précis qu'il s'occupera de mettre à jour les données d'une part, et les vues d'autre part.

Boîtes de dialogue

Les boîtes de dialogue sont essentielles et sont partie intégrante de l'interface utilisateur. Nous avons fait le choix d'utiliser des composants directement importés de la bibliothèque Swing. Ce choix a été fait pour des raisons d'efficacité, d'esthétique, et surtout du fait que Swing soit plus récent et maintenu à jour fréquemment qu'AWT.

Pour la disposition des composants, chose très fastidieuse à faire manuellement, nous avons fait le choix du *Layout* null. Cela permet de disposer et redimensionner à notre guise les composants.

Il est à noter que toutes les boîtes de dialogue spécialisent la classe *JDialog*. Cela dans le but de permettre une navigation plus intuitive et moins perturbante pour l'utilisateur, puisqu'en utilisant des fenêtres modales, on empêche la confusion.

La palette

La palette est un élément important pour la scène puisque grâce à elle l'utilisateur va pouvoir effectuer l'action qu'il souhaite :

+ Sélection : Permet de sélectionner les différents objets présents sur la scène
+ Classe : Permet la création d'une classe sur la scène
+ Agrégation/Composition : Lien d'association entre deux classes
+ Héritage : Lien de spécialisation entre deux classes
+ Note : Permet d'écrire une note d'information concernant le diagramme
+ Trait simple : Donne à l'utilisateur la possiblité de dessiner un trait sans point d'ancrage

Nous avons décidé de faire dériver la palette du composant JToolBar, une sorte de conteneur qui affiche ses composants à la manière d'une barre d'outils, verticalement ou horizontalement. L'idée était de laisser à l'utilsateur la possibilité de pouvoir placer la palette là où l'utilisateur le souhaitait.

Sauvegarde

La sauvegarde de diagrammes, ou de projets entiers nous paraissait nécessaire, ne serait-ce que pour permettre à l'utilisateur n'ayant pas terminé sa modélisation, de la reprendre le lendemain sans avoir besoin de laisser sa machine allumée.

Tout ce traitement de données a été géré en utilisant la sérialisation native proposée par Java. Elle permet la conservation dans des fichiers binaires d'informations ayant trait à des objets précis, et permet bien évidemment de les récupérer en chargeant le fichier.

La seule chose à faire pour sérialiser un objet est de lui faire implémenter l'interface *java.io.Serializable.*

Drag & Drop

Le principe est simple : il suffit de sélectionner à l'aide de la souris un élément extérieur, de l'amener au dessus de la fenêtre de l'application, puis de le relâcher. Cela fait partie des fonctionnalités simples et utiles que nous avons souhaité mettre en place.

Le drag & drop a été implanté pour réagir à trois types d'actions : ajouter une classe contenue dans l'arbre sur la scène, ouvrir des diagrammes, ou des projets complets, et importer des fichiers sources dans les différents langages supportés.

Le format des fichiers lâchés par drag & drop sont identifiés par des *DataFlavor*. Un *DataFlavor* n'est rien d'autre qu'un objet Java qui représente un type MIME (par exemple, un fichier html sera identifié comme : "text/html", un fichier image pourra être décrit comme "image/png", et un fichier pdf comme "application/pdf"). Cette classe contient d'ailleurs des constantes pour les types les plus courants.

Pendant les différentes phases de test du logiciel nous avons pu nous rendre compte que la prise en charge des types par la classe *DataFlavor* était différente selon le système d'exploitation. Il nous a donc fallu pallier ce problème pour conserver la cohérence et l'idée de portabilité.

4.2 L'exportation de code

L'exportation de code se base sur une classe nommée *GenerationCode*, et suit une structure XML précise mais souple, dont les présentations vont vous être faites.

4.2.1 La classe GenerationCode

Au départ, l'idée était que chaque objet d'un diagramme des classes (*Classe*, *Attribut*, *Méthode*, etc...) puisse savoir lui-même se générer seul. Cela induisait l'idée que chacun de ces objets comprenne

une méthode de génération. Mais cette solution ne nous a pas paru concluante. En effet, de part la conception, nous souhaitions appliquer le patron Modèle/Vue-Contrôleur, et nous avons donc trouvé mal à propos d'altérer le modèle en y ajoutant une méthode qui ne faisait que déservir l'idée et le rôle de base de ces objets.

Face à ce constat, il nous fallait trouver une autre solution. Nous avons donc choisi plus simplement de dédier une classe entière à la génération de chacun de ces objets : la classe *GenerationCode* était née. Cette classe a été volontairement sectionnée en deux parties : la partie objet pure, et la partie statique.

L'objet

En effet, la classe *GenerationCode*, bien que les constructeurs aient été mis volontairement en visibilité privée, est un objet à part entière. Pour créer l'objet il faut passer le chemin du fichier XML contenant la structure nécessaire à l'exportation, dans le but de le parser le fichier et d'initialiser les différents attributs.

Les méthodes statiques

La seconde partie de cette classe est caractérisée des méthodes statiques toutes de visibilité privée, sauf celle que l'on pourrait qualifier de principale : *genererDiagramme(DiagrammeClasses d, String fichierXML, String repertoireExportation)*. Celle-ci permet comme son nom l'indique, de générer un diagramme en suivant la structure XML contenue dans le fichier XML. Les autres méthodes sont toutes appelées, mais uniquement en interne. Notre idée était de ne laisser transparaître que les méthodes qui pourraient être indépendantes.

Il y a donc plusieurs catégories dans ces méthodes statiques. Certaines sont spécifiquement utilisées dans la génération de chacun des objets d'un diagramme (*genererClasse(...)*, *genererAttribut(...)*, *genererMethode(...)*, etc...) réagissant chacune à des mots-clefs précis, d'autres servent à générer des objets qui ne sont pas représentés sur un diagramme, mais qui sont nécessaires dans le code (*genererContructeur(...)* par exemple). Puis viennent ensuite des méthodes permettant d'effectuer des vérifications sur le type d'objet passé en paramètre (savoir si il respecte un certain filtre, correspond bien à certains critères, etc...), et enfin des méthodes permettant la mise en forme du code (*formaterCode(...)*, *tabulation(...)*).

4.2.2 Structure XML

La structure des fichiers XML permettant l'exportation de code dans un langage donné est relativement simple. Sa mise en place a toutefois été quelque peu difficile, car cela revenait à créer une sorte de norme qui se devait d'être souple vu les différences entre les langages existants. La structure, après un noeud père du nom de "langage" avec un attribut précisant son nom, se divise en trois parties : les mots-clefs, les déclarations d'objets et les fichiers.

Les mots-clefs

Cette partie de la structure est très importante puisqu'elle regroupe tous les mots-clefs que l'on souhaite utiliser dans le document. Il suffit de rajouter, entre les deux balises "keywords", des balises dont le nom sera utilisé dans la suite du document, et d'y insérer la valeur. Par exemple, si l'on écrit $< endl >;< /endl >$, à chaque fois que le mot-clef *endl* sera trouvé dans le fichier XML, c'est le point virgule qui sera inséré dans le code à la place.

Cette particularité permet entre autre de changer toute le code d'une exportation en ne changeant qu'un mot du fichier. Ce mécanisme est très souple puisqu'il est totalement adaptable en fonction des besoins de l'utilisateur.

Les déclarations d'objets

Viennent ensuite sept sections obligatoires (au moins vides) que sont : *constructeurs*, *heritages*, *agregations*, *attributs*, *paramètres*, *méthodes* et *classes*. Dans ces sections peuvent être insérées des sous-section dont le nom est laissé à l'appréciation de l'utilisateur. Ces sections permettent de déterminer la façon dont les objets seront générés. Chacun des attributs des objets est accessible à condition d'être dans la bonne section. Par exemple, si nous nous trouvons dans la partie concernant les méthodes nous pouvons avoir ceci :

$$< it > visibilite < /it >< it > espace < /it >< it > function < /it >< it > espace < /it >$$

$$< it > nom < /it >< it > (< /it >< parametres >< premier > premierParametre < /premier >$$

$$< autres > autresParametres < /autres >< /parametres >< it >) < /it >$$

Dans ce cas précis, les mots-clefs "visibilité", "nom" et "parametres" font appel aux attributs de la classe Méthode. A la génération le code retourné sera la valeur de ces attributs. L'astuce concernant les paramètres est appliquée dans la plupart des cas, puisqu'en fait à partir du moments ou l'on souhaite faire appel à un attribut de type liste, il suffit de mettre entre des balises du nom des objets que l'on souhaite récupérer, les pointeurs vers les sous-sections des objets de base : ici les paramètres seront déclarés en utilisant les sous-section "premierParametre" et "autresParametres" de la section *parametre*.

Les fichiers

La troisième est dernière partie du fichier est la plus importante. En effet, elle permet à l'utilisateur de déclarer entre deux balises *fichier* la façon dont il faut exporter les classes. Deux cas sont prévus.

Si la sous-section commence par la balise $< classesnom = "nomClasse"extension = ".php" >$, la génération se fera pour chacune des classes du diagramme dans un fichier dont le nom sera le nom de classe en cours, et l'extension ".php".

Le second cas pris en compte concerne la création de fichiers autres que ceux ayant directement trait aux classes. Par exemple, si l'on trouve $< autrenom = "Main"extension = ".cpp" >$, le code sera généré dans le fichier "Main.cpp".

La création de ce type de fichiers XML est développée plus en détails dans le manuel du logiciel.

4.3 Le reverse engineering

Le code Flex est divisé en trois parties : code utilisateur, options et déclarations, règles lexicales. Leur intérêt et leur implémentation sont détaillés ci-dessous.

4.3.1 Code utilisateur

Dans cette partie, il nous a fallu importer toutes les classes nécessaires à la création d'un diagramme (DiagrammeClasses, Méthode, Attribut, etc...) que nous utiliserons par la suite. Sans ces importations, impossible d'instancier des objets de ce type.

4.3.2 Options et déclarations

C'est ici que nous donnons le nom du fichier qui sera généré ainsi que diverses options. Nous déclarons également les fonctions qui pourront être utilisées dans la suite du code (par exemple la fonction reconnaitVar()).

Les principaux types d'expressions (sous forme d'expression régulière), permettant de reconnaître les blancs, les bibliothèques importées, les commentaires ou encore les identifiants sont eux-aussi déclarés ici.

Il nous a fallu être très rigoureux et parcourir la documentation à notre disposition pour utiliser au maximum les possibilités offertes par JFlex, car certains expressions peuvent être reconnues par plusieurs expressions régulières, mais toutes ne sont pas optimisées par le compilateur créant l'automate.

4.3.3 Règles lexicales

C'est dans cette partie du code que nous écrivons toutes les règles qui nous intéressent pour le type de fichier scanné (Java, C++, etc...). Ainsi, pour chaque règle inscrite, le code permettant de récupérer les éléments qui nous intéressent (description, type et nom d'un attribut par exemple) est bien évidemment différent. A chaque fois qu'un élément est analysé et reconnu, il est récupéré pour être inséré dans la classe correspondante.

4.3.4 Génération du source Java

Une fois que le code flex finalisé, nous n'avions plus qu'à générer le code source Java correspondant, grâce à la commande *jflex fichier.flex*.

Chapitre 5

Perspectives & Conclusions

Nous voici enfin à l'heure du bilan concernant le déroulement de ce projet. Qu'en est-il réellement de son ergonomie, de l'idée que nous nous en étions faite, et bien sûr, qu'en adviendra-t-il dans l'avenir ? Nous essayons également d'analyser notre travail de groupe, dans le but de mettre en évidence les points positifs et négatifs.

5.1 Perspectives

Ce projet ayant atteint la phase finale de sortie de la version 1.0, il nous faut penser à son avenir. Il est vrai que la plupart des logiciels conçus lors de TER à l'université sont souvent abandonnés dès la soutenance. Mais si cela est vrai, c'est en grande partie dû à la façon d'aborder le sujet, ainsi qu'au sujet en lui-même...

Le projet semble être ici très différent, puisque le sujet est une libre proposition de notre part. Nous avons nous-mêmes réfléchi à nos besoins quotidiens en tant qu'étudiants en informatique, qui conceptualisons, qui programmons, et nous avons réussi à trouver une certaine lacune de logiciels libres, gratuits, bien faits, et correspondant à nos besoins dans le domaine de la modélisation UML.
Etant donc à la source de ce projet du début à la fin, nous avons réellement pris en compte notre propre désir de voir réunies les meilleures fonctionnalités envisageables dans un tel logiciel, et de ce fait, nous utiliserons ce logiciel pour nos études et nos projets futurs.

Nous sortirons sûrement des versions plus abouties, que ce soit dans l'hypothèse où des idées de fonctionnalités manquantes nous apparaîtraient, ou si des bugs venaient à être découverts.

5.2 Conclusions

Que pensons-nous du résultat final obtenu ? Qu'avons-nous pensé de l'alchimie du travail de groupe ?

5.2.1 Critique de l'application

Il est certain que réussir à critiquer ouvertement et objectivement le fruit de nos propres mains est une tâche ardue, mais il s'avère tout de même nécessaire de le faire.

Globalement parlant, le résultat de nos trois mois de travail acharné (surtout sur la fin...), nous apparaît comme très intéressant.

La prouesse technique d'avoir réussi à domestiquer Swing au point de réussir à permettre la modélisation de diagrammes reste une de nos grandes satisfactions. En effet, ce volet, qui comptait parmi les fondations de notre logiciel, ne nous était pas acquis dès le départ, puisque nous avions l'envie, mais pas forcément les connaissances nécessaires pour maîtriser la programmation de cette partie.

D'autre part, le travail effectué sur la partie concernant l'importation de classes a été également très enrichissante et dans la droite lignée de l'application de notre cours d'Interprétation & Compilation, avec l'approfondissement de certains points en prime. Le résultat est tout à fait à la hauteur de notre espérance de départ, d'autant plus que nous avions dans un premier temps décidé d'avancer la conception de base avant d'être sûrs de nous lancer corps et âme dans l'approche et la réalisation du reverse engineering.

Le troisième point qu'il nous semble essentiel de mettre en avant puisqu'il faisait partie intégrante des fonctionnalités de départ décrites dans le cahier des charges, c'est celui de l'exportation de code. Ce module a toute son importance de part sa conception totalement externalisée puisqu'elle suit un schéma contenu dans un fichier de type XML. Il réussit un pari que nous nous étions faits, qui comptait sur la portabilité et l'extension des possibilités du logiciel, de par la possibilité ultérieure d'ajout de nouveaux langages pour l'exportation par de futurs utilisateurs.

Viennent maintenant les points négatifs de notre application. En effet, comme toute application, et d'ailleurs comme toutes les premières versions, elle n'est pas exempte de défauts. Ils sont la plupart du temps peu importants puisque nous avons réussi à canaliser nos efforts en essayant de réaliser le mieux possible les fonctionnalités de base plutôt que d'en faire le double à moitié, mais existent quand même.

Un de points que nous souhaiterions améliorer dans le futur, est l'ergonomie des boîtes de dialogue de création et de modification des objets du diagramme. Elles peuvent être repensées et améliorées pour réussir une meilleure intégration et accroître la facilité et la rapidité de création.

L'autre lacune importante reste celle de l'abandon de notre part (pour des raisons de planning), reste l'impossibilité de faire des diagrammes UML autres que les diagrammes de classes. Même si ce type de diagramme représente une partie essentielle, et sûrement celle qui fait la jointure la plus vitale avec la partie programmation, UML comprend d'autres types de diagramme ayant chacun leur intérêt.

Ces défauts, sans chercher de justification bancale et/ou hâtive, sont en partie malheureusement dus au peu de temps de réalisation que nous avons eu, chose que nous déplorons beaucoup, mais apprécions également, car travailler dans une contrainte de temps reste toujours un défi, et demande une grande efficacité et des prises de décisions et de risques rapides, mais mesurées.

5.2.2 Travail au sein du groupe

Le travail en équipe a été rudement mené. L'équipe ne se connaissait que depuis le début de l'année scolaire, mais l'alchimie fut très bonne. Le fait, encore une fois, d'avoir proposé nous-mêmes le sujet et l'équipe qui travaillerait dessus, nous a donné un certain avantage dans la collaboration. Nous sommes la plupart du temps dans les mêmes groupes de TD/TP, ce qui nous a permis d'avoir un contact permanent, et d'éviter les réunions hebdomadaires souvent difficiles à organiser en fonction des obligations de chacun.

Il faut savoir que comme nous l'avons déjà précisé, l'équipe s'était scindée en deux groupes, Gabriel & Guillaume travaillant sur la partie graphique de l'application, Laurent & Fabien s'occupant respectivement du reverse engineering et de l'exportation de code. Etant donné que Fabien était le chef de projet, les prises de décisions majeures étaient bien évidemment effectuée avec son accord.

Il est cependant bon de noter que Gabriel a quelque fois pris les rennes pour diriger les décisions graphiques, pour des raisons d'efficacité, et parce que l'équipe avait confiance en ses choix.

Bibliographie

[Cahier du programmeur Swing] Emmanuel Puybaret, *Swing, Eyrolles, 2006*

[Documentation Java] *http ://java.sun.com/j2se/1.4.2/docs/api/*

[Java Reference] *http ://www.javareference.com/*

[Java examples] *http ://www.java2s.com/*

[JavaGaming forum] *http ://www.javagaming.org/forums/*

[Ressources de développement] *http ://www.developpez.com/*

[Encyclopédie collaborative] *http ://fr.wikipedia.org/wiki/Accueil*

[Ressources Java] *http ://www.javafr.com/*

[JFlex - The Fast Scanner Generator for Java] *http ://www.jflex.de/*

Annexe A

Dossier d'analyse



FIG. A.1 – Diagramme des cas d'utilisation

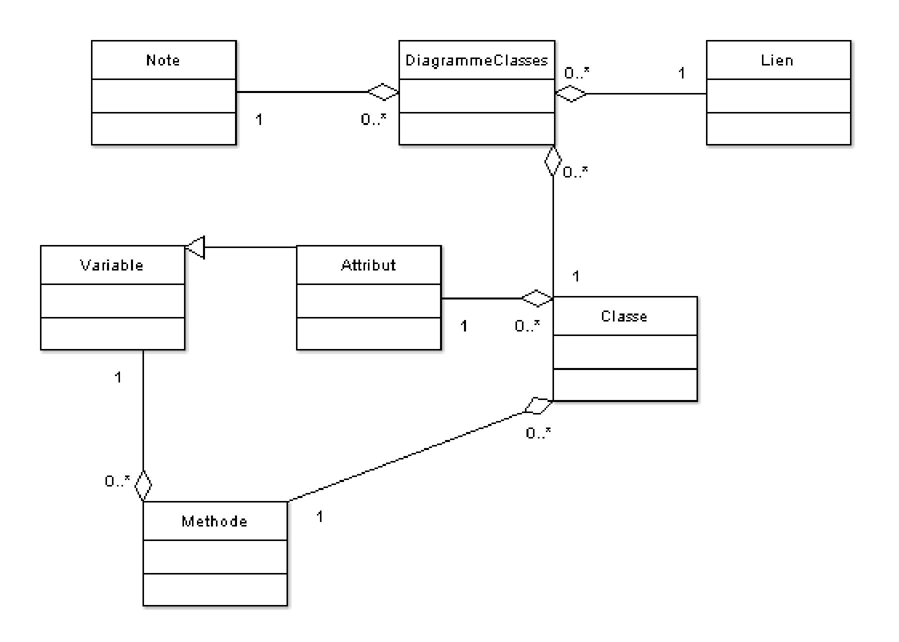


FIG. A.2 – Diagramme de classes de la partie Noyau

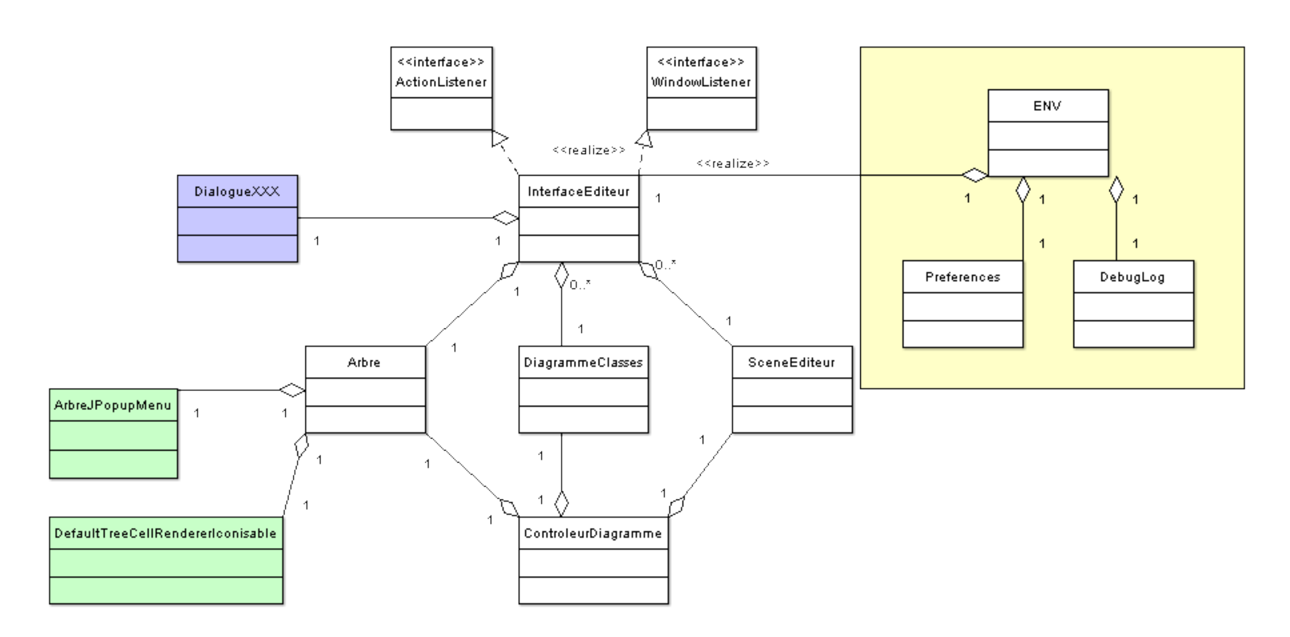


FIG. A.3 – Diagramme de classes de l'interface du logiciel

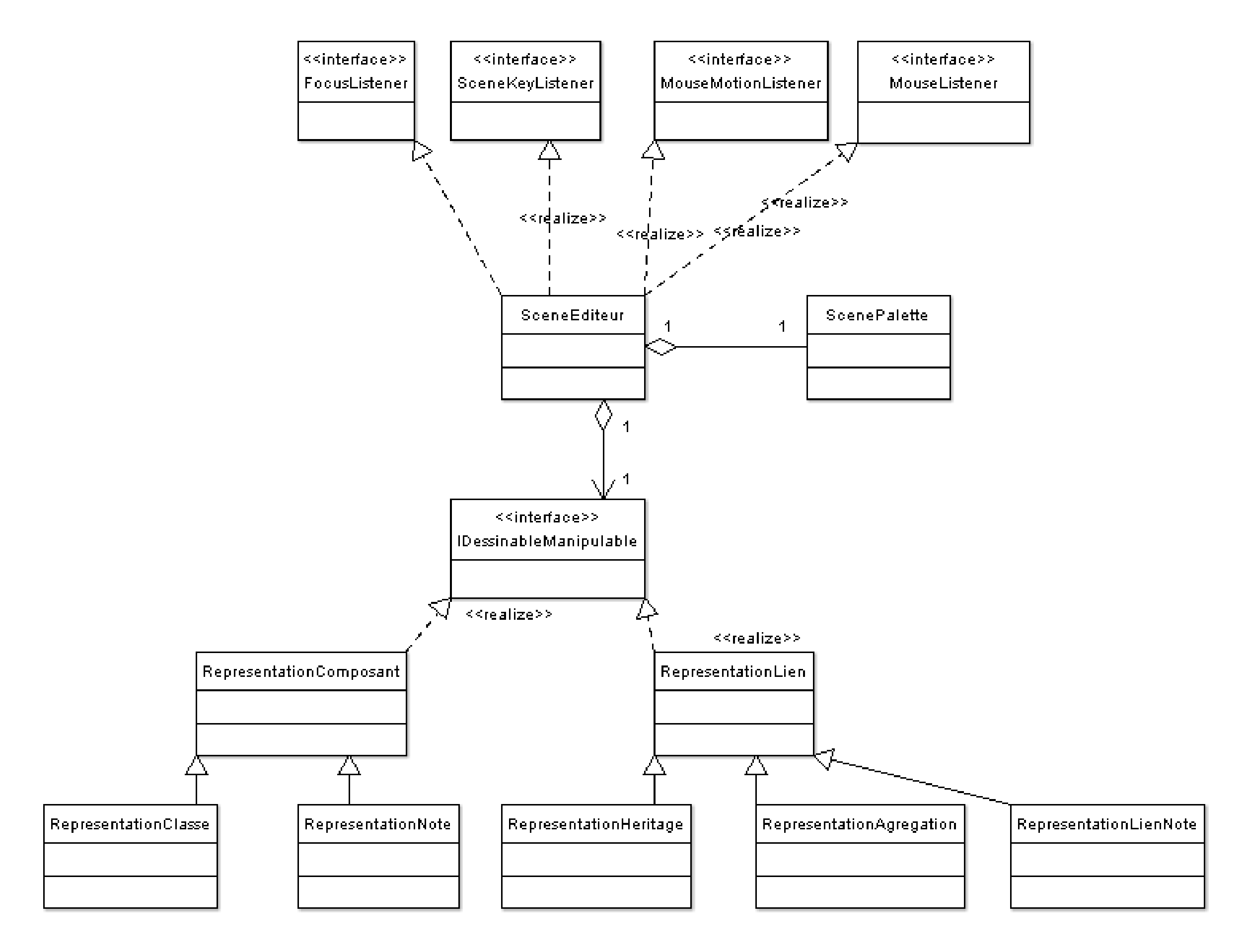


Fig. A.4 – Diagramme de classes de la scène

Annexe B

Extraits de programmation

B.1 Une boîte de dialogue

L'interface passe sans conteste par l'utilisation de boîtes de dialogue permettant l'interaction avec l'utilisateur. Celle qui suit permet à l'utilisateur de sélectionner certains fichiers destinés à être importés dans le logiciel via la fonction de reverse engineering.

```
package gui.dialogues;

import [...]


/**
 * Boite de dialogue permettant de sélectionner les préférences
 * pour le reverse engineering
 * @author Dalichoux Guillaume
 */
public class DialogueReverseEng extends JDialog implements ActionListener {

    /** Boutons **/
    protected JButton btImporter = new JButton("Importer");
    protected JButton btAnnuler = new JButton("Annuler");
    protected JButton btParcourir = new JButton("Parcourir");
    protected JButton btSelectAll = new JButton("Toutes");
    protected JButton btDeselectAll = new JButton("Aucune");

    /** Labels et Composants **/
    protected JLabel lblListeClasses;
    private JCheckBoxList cblListeClasses;

    /** Autres **/
    protected DiagrammeClasses diagramme;
    protected ArrayList<String> listeCheminClasses;
    protected ArrayList<JCheckBox> listeJCB;


    /**
     * Constructeur par défaut
     * @param owner Fenêtre parente
     * @param diagramme Diagramme auquel il faut ajouter les classes importées
     */
    public DialogueReverseEng(JFrame owner, DiagrammeClasses diagramme) {
```

```
    super(owner, true);
    this.diagramme = diagramme;
    this.listeCheminClasses = new ArrayList<String>();
    this.listeJCB = new ArrayList<JCheckBox>();
    initialize();
}


/**
 * Permet d'initialiser la fenêtre
 */
private void initialize() {
    // On donne un titre  l'application
    this.setTitle(" Importation de classes ");
    // On donne une taille  notre fentre
    this.setSize(392,340);
    // On ne permet pas le redimensionnement
    this.setResizable(false);
    // On dit  l'application de se fermer lors du clic sur la croix
    this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

    // panneau de fond
    JPanel conteneur = new JPanel();
    conteneur.setBackground(new Color(230,230,230));
    conteneur.setLayout(null);

    // panneau de génération
    JPanel panneauGenerer = new JPanel();
    panneauGenerer.setBackground(new Color(243, 243, 243));
    panneauGenerer.setBorder(BorderFactory.createLineBorder(Color.black));
    panneauGenerer.setBounds(5,5,375,270);
    panneauGenerer.setLayout(null);

    // liste des classes  cocher
    JLabel titreListe = new JLabel("Cochez les classes à importer :");
    titreListe.setBounds(10, 10, 200, 16);
    panneauGenerer.add(titreListe);
    this.cblListeClasses = new JCheckBoxList();
    this.cblListeClasses.setBorder(BorderFactory.createLineBorder(Color.black));
    this.cblListeClasses.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    JScrollPane scrollPane = new JScrollPane(cblListeClasses);
    scrollPane.setBounds(10, 35, 235, 220);
    panneauGenerer.add(scrollPane,cblListeClasses);

    // boutons
    btSelectAll.setBounds(255, 35, 110, 22);
    btSelectAll.addActionListener(this);
    panneauGenerer.add(btSelectAll);

    btDeselectAll.setBounds(255, 65, 110, 22);
    btDeselectAll.addActionListener(this);
    panneauGenerer.add(btDeselectAll);

    btParcourir.setBounds(255, 95, 110, 22);
    btParcourir.addActionListener(this);
    panneauGenerer.add(btParcourir);
```

```
    btImporter.setBounds(190, 285, 90, 22);
    btImporter.addActionListener(this);
    conteneur.add(btImporter);

    btAnnuler.setBounds(290, 285, 90, 22);
    btAnnuler.addActionListener(this);
    conteneur.add(btAnnuler);

    conteneur.add(panneauGenerer);
    setContentPane(conteneur);

    // banalités d'usage sur une fenêtre
    this.setLocationRelativeTo(null);
    this.setVisible(true);
}


/**
 * Renvoie le répertoire choisi par l'utilisateur via une boite de dialogue
 */
private File getRepertoire() {
    File dossier = new File("");
    // on créé un JFileChooser
    JFileChooser jfc = new JFileChooser();
    // on positionne le répertoire
    jfc.setCurrentDirectory(new java.io.File(""));
    // on met un titre
    jfc.setDialogTitle("Sélection du dossier d'importation");
    // on force le jfc n'accepter que les répertoires
    jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    // on enlève tous les filtres de fichiers
    jfc.setAcceptAllFileFilterUsed(false);
    // si tout va bien  la fermeture...
    if (jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
        // on récupère le path absolu
        dossier = jfc.getSelectedFile();
    }
    else
    {
        System.err.println("Aucun répertoire sélectionné...");
    }
    return dossier;
}


/**
 * Permet de scanner les fichiers différents fichiers contenus dans le répertoire
 * @param rep Répertoire à scanner
 * @throws IOException
 */
private void scanRepertoire(File rep)
{
    if (rep.isDirectory()) {
        // on créé le répertoire
        File repertoire = rep;
```

```
        // on récupère la liste des fichiers dans un tableau
        File []  listeFichiers = repertoire.listFiles();
        // pour chacun des fichiers de la liste
        for (File f : listeFichiers) {
            if (f.isDirectory()) {
                scanRepertoire(f);
            }
            else {
                String nomFichier = f.getName();
                if (fichierReconnu(nomFichier)) {
                    String cheminCompletFichier = "";
                    try {
                        cheminCompletFichier =
                            DialogueGeneration.path(repertoire.getCanonicalPath())
                            + nomFichier;
                    } catch (IOException e) {
                        System.err.println("Erreur sur le fichier " + nomFichier);
                    }
                    if (!cheminCompletFichier.isEmpty()) {
                        this.listeJCB.add(new JCheckBox(nomFichier));
                        System.out.print("\""+cheminCompletFichier+"\" ");
                        this.listeCheminClasses.add("\""+cheminCompletFichier+"\"");
                    }
                }
            }
        }
    }
}


/**
 * Renvoie VRAI si le fichier passé en paramètre est reconnaissable par notre fonction de Reverse
 * @param fichier Fichier à tester
 */
private boolean fichierReconnu(String fichier) {
    return (fichier.endsWith(".h") || fichier.endsWith(".cc") || fichier.endsWith(".cpp") ||
        fichier.endsWith(".java") || fichier.endsWith(".php") || fichier.endsWith(".py"));
}


/**
 * Si une action se produit...
 */
public void actionPerformed(ActionEvent e) {
    // on récupère la source
    Object ao = e.getSource();
    // si clic sur le bouton Importer
    if (ao.equals(this.btImporter)) {
        ArrayList<String> cheminsClasses = new ArrayList<String>();
        for (int i = 0; i < this.listeJCB.size(); i++) {
            if (this.listeJCB.get(i).isSelected()) {
                cheminsClasses.add(this.listeCheminClasses.get(i));
            }
        }
        try
        {
```

```
            this.diagramme.concatenerDiagramme(Scanne.Scan(cheminsClasses));
            }
            catch(NullPointerException enull) {}
            this.dispose();
        }
        // si clic sur le bouton Annuler, on ferme la fenêtre
        else if (ao.equals(this.btAnnuler)) {
            this.dispose();
        }
        // si clic sur le bouton Parcourir, on fait choisir un répertoire que l'on analyse aussitôt
        else if (ao.equals(this.btParcourir)) {
            scanRepertoire(getRepertoire());
            this.cblListeClasses.setListData(this.listeJCB.toArray());
        }
        // clic sur le bouton Toutes
        else if (ao.equals(this.btSelectAll)) {
            for (JCheckBox cb : this.listeJCB) {
                cb.setSelected(true);
            }
            this.cblListeClasses.setListData(this.listeJCB.toArray());
        }
        // clic sur le bouton Aucune
        else if (ao.equals(this.btDeselectAll)) {
            for (JCheckBox cb : this.listeJCB) {
                cb.setSelected(false);
            }
            this.cblListeClasses.setListData(this.listeJCB.toArray());
        }
    }


}
```

B.2 La scène

B.2.1 Classe RepresentationComposant

Cette classe est abstraite et implémente l'interface IDessinableManipulable (nécessaire pour un objet destiné à appartenir à la scène). Elle sera elle-même nécessaire pour créer un composant à dessiner sur la scène, car elle possède toutes les propriétés obligatoires (sa position, ses dimensions, etc...). Il faudra bien évidemment penser à redéfinir la méthode *void dessiner(Graphics2D g2d)* qui permettra de dessiner le nouveau composant.

```
package scene.composants;

import [...]


/**
 * Composant Graphique
 * Chaque composant peut dériver de RepresentationComposant
 * La méthode void dessiner(Graphics2D g2d) doit être redefinie
 * Chaque composant possède son affichage et ses propriétés
 * @author Delmas Gabriel
 */
public abstract class RepresentationComposant implements IDessinableManipulable, Serializable {
```

```
/** Constante largeur minimum d'un composant **/
protected final int MIN_LARGEUR=100;

/** Constante hauteur minimum d'un composant **/
protected final int MIN_HAUTEUR=100;

/** Largeur minimum du composant **/
protected int minLargeur=100;

/** Hauteur minimum du composant **/
protected int minHauteur=100;

/** Largeur du composant **/
protected int largeur=100;

/** Hauteur du composant **/
protected int hauteur=100;

/** Position (coin haut gauche) du composant **/
protected Point p;

/** Dimensions du composant (sa surface) **/
protected Rectangle cadre;

/** Taille en pixel de l'espacement des écritures **/
protected int espace_texte=12;

/** Permet de savoir si le composant est selectionné **/
protected boolean estSelectionne;

/** Permet de savoir si le composant est en train d'être déplacé **/
protected boolean estTransfere;

/** Permet de savoir si l'ombrage du composant est activé **/
protected boolean ombreComposant;

/**
 * Les 4 extremités du composants représentent la selection du composant et permet d'agir sur
 * les dimensions du composants
 **/
protected PointExtremite pext_hd, pext_hg, pext_bd, pext_bg;

/** Nom de l'extrémité sélectionnée **/
protected String extremiteSelectionnee;

/** Permet de garder le centrage de la figure lors du déplacement **/
private Dimension xyespace;

/** Couleur de fond du composant **/
protected Color couleurFond;

/** Couleur de la bordure du composant **/
protected Color couleurBordure;

/** Couleur du texte du composant **/
```

```
protected Color couleurTexte;


/**
 * Constructeur prenant en paramètre la position du composant
 * @param p la position du composant (coin haut gauche)
 */
public RepresentationComposant(Point p)
{
    this.p=p;
    this.estSelectionne=false;
    this.estTransfere=false;
    this.ombreComposant=false;
    this.extremiteSelectionnee="";
    xyespace = new Dimension(0,0);

    //Couleur de bases
    couleurFond=new Color(238, 238, 238);
    couleurBordure=Color.black;
    couleurTexte=Color.black;

    //On calcule les points des extremites
    pext_hg=new PointExtremite(p.x,p.y);
    pext_hd=new PointExtremite(p.x+largeur,p.y);
    pext_bg=new PointExtremite(p.x,p.y+hauteur);
    pext_bd=new PointExtremite(p.x+largeur,p.y+hauteur);

    //On definit les dimensions du composants
    cadre=new Rectangle(p.x,p.y,largeur,hauteur);
}


/**
 * Calcul des dimensions du composant et du positionnement des extrémités
 */
protected void CalculGraphique() {
    Point pPos;
    //On calcule les points des extremites
    pPos=new Point(p.x,p.y);
    pext_hg.setPosition(pPos);
    pPos=new Point(p.x+largeur,p.y);
    pext_hd.setPosition(pPos);
    pPos=new Point(p.x,p.y+hauteur);
    pext_bg.setPosition(pPos);
    pPos=new Point(p.x+largeur,p.y+hauteur);
    pext_bd.setPosition(pPos);
    cadre.setBounds(p.x,p.y,largeur,hauteur);
}


/**
 * Méthode abstraite à implementer
 * Affichage graphique du composant (chaque composant possède son propre affichage)
 */
public abstract void dessiner(Graphics2D g2d);
```

```
[...]

/**
 * Désélection des 4 extrémités
 */
public void relacherExtremites() {
    pext_hg.setSelection(false);
    pext_hd.setSelection(false);
    pext_bg.setSelection(false);
    pext_bd.setSelection(false);
}


/**
 * Vérifie si un point est contenu dans une des extrémités
 * @param Point correspondant à un clic
 * @return boolean si le point est contenu dans une des extrémités
 */
public boolean intersectionExtremite(Point p) {
    if(pext_hg.Clique(p)) {
        extremiteSelectionnee="haut_gauche";
        pext_hg.setSelection(true);
    }
    else if(pext_hd.Clique(p)) {
        extremiteSelectionnee="haut_droite";
        pext_hd.setSelection(true);
    }
    else if(pext_bg.Clique(p)) {
        extremiteSelectionnee="bas_gauche";
        pext_bg.setSelection(true);
    }
    else if(pext_bd.Clique(p)) {
        extremiteSelectionnee="bas_droite";
        pext_bd.setSelection(true);
    }
    return pext_hg.getSelection()||pext_hd.getSelection()||pext_bg.getSelection()
        ||pext_bd.getSelection();
}


/**
 * Déplace l'extrémité sélectionnée au préalable par intersectionExtremite()
 * Permet de modifier les dimensions d'un composant suivant le déplacement d'une extrémité
 * @param Point contenant les coordonnées de destination
 */
public void deplacerExtremite(Point pnouveau) {
    if(extremiteSelectionnee=="haut_gauche") {
        if(largeur-(pnouveau.x-this.p.x)>=minLargeur) {
            largeur-=(pnouveau.x-this.p.x);
            this.p.x+=(pnouveau.x-this.p.x);
        }
        if(hauteur-(pnouveau.y-this.p.y)>=minHauteur) {
            hauteur-=(pnouveau.y-this.p.y);
            this.p.y+=(pnouveau.y-this.p.y);
```

```
        }

    }
    else if(extremiteSelectionnee=="haut_droite") {
        if(largeur+(pnouveau.x-this.p.x)-largeur>=minLargeur) {
            largeur+=(pnouveau.x-this.p.x)-largeur;
            this.p.x+=(pnouveau.x-this.p.x-largeur);
        }
        if(hauteur-(pnouveau.y-this.p.y)>=minHauteur) {
            hauteur-=(pnouveau.y-this.p.y);
            this.p.y+=(pnouveau.y-this.p.y);
        }
    }
    else if(extremiteSelectionnee=="bas_gauche") {
        if(largeur-(pnouveau.x-this.p.x)>=minLargeur) {
            largeur-=(pnouveau.x-this.p.x);
            this.p.x+=(pnouveau.x-this.p.x);
        }
        if(hauteur+(pnouveau.y-this.p.y)-hauteur>=minHauteur) {
            hauteur+=(pnouveau.y-this.p.y)-hauteur;
            this.p.y+=(pnouveau.y-this.p.y)-hauteur;
        }
    }
    else if(extremiteSelectionnee=="bas_droite") {
        if(largeur+(pnouveau.x-this.p.x)-largeur>=minLargeur) {
            largeur+=(pnouveau.x-this.p.x)-largeur;
            this.p.x+=(pnouveau.x-this.p.x-largeur);
        }
        if(hauteur+(pnouveau.y-this.p.y)-hauteur>=minHauteur) {
            hauteur+=(pnouveau.y-this.p.y)-hauteur;
            this.p.y+=(pnouveau.y-this.p.y-hauteur);
        }
    }
    CalculGraphique();
}


/**
 * Verifie si un Point est contenu dans l'objet
 * @param Point correspondant à un clic
 */
public boolean intersection(Point p) {
    Rectangle rect_select = new Rectangle(cadre);
    rect_select.x-=3;
    rect_select.y-=3;
    rect_select.width+=6;
    rect_select.height+=6;
    if(rect_select.contains(p)) return true;
    else return false;
}


/**
 * Vérifie si un rectangle entrecoupe le composant
 * @param Rectangle correspondant à un clic
 */
```

```
    public boolean intersection(Rectangle r) {
        Rectangle rect_select = new Rectangle(cadre);
        rect_select.x-=3;
        rect_select.y-=3;
        rect_select.width+=6;
        rect_select.height+=6;
        if(rect_select.intersects(r)) return true;
        else return false;
    }


    [...]


}
```

B.2.2 Classe RepresentationClasse

Voici donc le composant héritant de RepresentationComposant qui permet de représenter une classe sur la scène. La méthode *void dessiner(Graphics2D g2d)* a été redéfinie. D'autres propriétés ont par ailleurs été ajoutées à ce composant, comme l'affichage de lignes de texte.

```
package scene.composants;

import [...]


/**
 * Composant Graphique représentant une classe (UML Diagramme des classes)
 * @author Delmas Gabriel
 */
public class RepresentationClasse extends RepresentationComposant {

    private static final long serialVersionUID = 816894361515087819L;

    /** Model de classe graphique **/
    private Classe classeModel;

    /** Ligne séparatrice du titre et des attributs **/
    private Ligne separateurT;

    /** Ligne séparatrice des attributs et des méthodes **/
    private Ligne separateurAM;

    /** Police d'écriture **/
    private FontMetrics fontMetrics;

    /** Affichage du contenu **/
    private boolean afficherContenu;


    /**
     * Constructeur
     * @param p la position (haut gauche)
     */
    public RepresentationClasse(Point p) {
```

```
    super(p);
    largeur=120;
    hauteur=150;

    afficherContenu=true;

    Point pseperateur_hg=new Point(p.x,p.y+20);
    Point pseperateur_bd=new Point(p.x+largeur,p.y+20);

    separateurT=new Ligne(pseperateur_hg, pseperateur_bd);

    pseperateur_hg=new Point(p.x,p.y+40);
    pseperateur_bd=new Point(p.x+largeur,p.y+40);
    separateurAM=new Ligne(pseperateur_hg, pseperateur_bd);

    CalculGraphique();
}


/**
 * Permet de lier la classe graphique à son modèle
 * @param classeModel le model
 */
public void ajouterModel(Classe classeModel) {
    this.classeModel= classeModel;
    CalculGraphique();
}

public Classe getModel(){return this.classeModel;}

public void setAfficherContenu(boolean affiche) {
    if(!affiche) {
        hauteur=MIN_HAUTEUR;
        largeur=MIN_LARGEUR;
    }
    this.afficherContenu=affiche;
    CalculGraphique();
}

public boolean getAfficherContenu(){return this.afficherContenu;}


/**
 * Calcul des dimensions et des positions des séparateurs
 */
protected void CalculGraphique() {
    int hauteurNom,hauteurAttributs,hauteurMethodes;
    if(afficherContenu) {
        hauteurNom=espace_texte;
        if(this.classeModel!=null) {
            if(this.classeModel.getAttributs().size()!=0)
                hauteurAttributs=espace_texte*this.classeModel.getAttributs().size();
            else
                hauteurAttributs=espace_texte;
        }
        else
```

```
            hauteurAttributs=espace_texte;
        if(this.classeModel!=null) {
            if(this.classeModel.getMethodes().size()!=0)
                hauteurMethodes=espace_texte*this.classeModel.getMethodes().size();
            else
                hauteurMethodes=espace_texte;
        }
        else
            hauteurMethodes=espace_texte;

        int newhauteur=hauteurNom+hauteurAttributs+hauteurMethodes+espace_texte;

        if(newhauteur>hauteur) {
            hauteur=newhauteur;
            minHauteur=newhauteur;
        }
    }
    else {
        hauteurNom=espace_texte;
        hauteurAttributs=espace_texte;
        hauteurMethodes=espace_texte;
    }

    hauteurNom+=5;
    hauteurAttributs+=5;
    hauteurMethodes+=5;

    super.CalculGraphique();

    Point pseperateur_hg=new Point(p.x,p.y+hauteurNom);
    Point pseperateur_bd=new Point(p.x+largeur,p.y+hauteurNom);

    separateurT=new Ligne(pseperateur_hg, pseperateur_bd);

    pseperateur_hg=new Point(p.x,p.y+hauteurNom+hauteurAttributs);
    pseperateur_bd=new Point(p.x+largeur,p.y+hauteurNom+hauteurAttributs);
    separateurAM=new Ligne(pseperateur_hg, pseperateur_bd);

}

/**
 * Affichage graphique du composant
 * @param g2d contexte graphique
 */
public void dessiner(Graphics2D g2d) {
    fontMetrics = g2d.getFontMetrics();
    Color originalCouleur=g2d.getColor();
    Composite originalComposite = g2d.getComposite();

    // Trace de l'ombre
    if(ombreComposant) dessinerOmbre(g2d);
    g2d.setComposite(originalComposite);
    g2d.setColor(originalCouleur);

    // Si le composant est en train d'être déplacé on ajoute l'effet de transparence
    if(estTransfere) g2d.setComposite(makeComposite(0.5F));
```

```
// Trace de l'arrière plan du composant
g2d.setColor(couleurFond);
g2d.fillRect(cadre.x,cadre.y,cadre.width,cadre.height);

// Trace du contour du composant
g2d.setColor(couleurBordure);
g2d.drawRect(cadre.x,cadre.y,cadre.width,cadre.height);

// Trace des séparateurs
separateurT.dessiner(g2d);
separateurAM.dessiner(g2d);
g2d.setColor(couleurTexte);

// On écrit le contenu de la classe, chaque ligne de texte est raccourcie en fonction de
// la largeur du composant
String sTemp;
sTemp = racourcirString(classeModel.getNom(),largeur);
g2d.drawString(sTemp, p.x+espace_texte, p.y+espace_texte);

if(afficherContenu) {
    int i=1;
    for (Iterator<Attribut> it = this.classeModel.getAttributs().iterator();
            it.hasNext();) {
        Attribut attribut = it.next();
        sTemp = racourcirString(attribut.toString(),largeur);
        g2d.drawString(sTemp, separateurT.getPosition().x+espace_texte,
            separateurT.getPosition().y+espace_texte*i);
        i++;
    }
    i=1;
    for (Iterator<Methode> it = this.classeModel.getMethodes().iterator(); it.hasNext();) {
        Methode methode = it.next();
        sTemp = racourcirString(methode.toString(),largeur);
        g2d.drawString(sTemp, separateurAM.getPosition().x+espace_texte,
            separateurAM.getPosition().y+espace_texte*i);
        i++;
    }
}
// On affiche un symbole pour que visuellement on puisse voir que l'affichage du contenu de la
// classe a été désactivé
else {
    Point pointT=new Point(separateurT.getPDestination().x-20,
        separateurT.getPDestination().y+9);
    Triangle triangleA=new Triangle(pointT, true);
    triangleA.dessiner(g2d);
}

// Trace des extrémités
if(estSelectionne) {
    pext_hg.dessiner(g2d);
    pext_hd.dessiner(g2d);
    pext_bg.dessiner(g2d);
    pext_bd.dessiner(g2d);
}
```

```
        // Valeurs originales
        g2d.setColor(originalCouleur);
        g2d.setComposite(originalComposite);

    }


    /**
     * Affichage graphique de l'ombre du composant
     * @param g2d contexte graphique
     */
    public void dessinerOmbre(Graphics2D g2d) {
        Composite originalComposite = g2d.getComposite();
        //Initialisaton de la transparence
        g2d.setComposite(makeComposite(0.5F));
        //darkGray pour l'effet
        g2d.setColor(Color.darkGray);
        g2d.drawLine(cadre.x,cadre.y+cadre.height+1,cadre.x+cadre.width+2,cadre.y+cadre.height+1);
        g2d.drawLine(cadre.x+cadre.width+1,cadre.y+cadre.height,cadre.x+cadre.width+1,cadre.y);
        //gray pour accentuer l'effet
        g2d.setColor(Color.gray);
        g2d.drawLine(cadre.x,cadre.y+cadre.height+2,cadre.x+cadre.width+2,cadre.y+cadre.height+2);
        g2d.drawLine(cadre.x+cadre.width+2,cadre.y+cadre.height,cadre.x+cadre.width+2,cadre.y);
        g2d.setComposite(originalComposite);
    }


    /**
     * Permet de raccourcir un String suivant une largeur en pixel
     */
    private String racourcirString(String s, int largeurPixel) {
        String sTemp;
        String sFinal="";
        for(int j=0; j<=s.length(); j++) {
            sTemp=s.substring(0, j);
            if(fontMetrics.stringWidth(sTemp)<largeur-espace_texte)
                sFinal=sTemp;
        }
        return sFinal;
    }


    /**
     * Double clic sur la classe
     * Permet d'editer la classe
     */
    public void actionDoubleClique() {
        ENV.FRAME.getControleur().modifierClasse(this);
    }


    /**
     * Affiche le menu contextuel du clic droit
     * Permet de visualiser les actions possibles sur la classe
     */
    public void actionMenuContextuel(Point p) {
```

```
        ENV.FRAME.getControleur().afficherMenuContextuel(this, p);
    }


}
```

B.2.3 Classe SceneEditeur

Voici donc la fameuse scène de l'interface. Elle hérite de JPanel (composant Swing de base), ce qui permet d'y placer des composants. En redéfinissant la méthode *void paintComponent(Graphics g)* et en créant nos propres composants graphiques (RepresentationClasse, RepresentationNote, etc...), notre scène prend forme, avec des objets tous manipulables.

```
package scene;

import [...]


/**
 * Vue graphique d'un diagramme des classes : scène où l'on va afficher tous les composants
 * @author Delmas Gabriel
 */
public class SceneEditeur extends JPanel implements Serializable, ComponentImprimable {

    private static final long serialVersionUID = -4485358736461323246L;

    /** Permet de savoir si le transparence des composants est activée sur toutes les scènes **/
    public static boolean activerTransparence=true;

    /** Permet de savoir si l'ombrage des composants est actif sur toutes les scènes **/
    public static boolean activerOmbrage=true;

    /** Couleur par défaut du fond des composants **/
    public static Color couleurComposantFond=new Color(238, 238, 238);

    /** Couleur par défaut de la bordure des composants **/
    public static Color couleurComposantBordure=Color.black;

    /** Couleur du texte par défaut des composants **/
    public static Color couleurComposantTexte=Color.black;

    /**
     * Palette de la scène, permet de sélectionner différents modes d'actions
     * Permet de sélectionner des outils pour interargir différemment sur la scène
     */
    private ScenePalette palette=null;

    /** Rectangle de sélection **/
    private RectangleSelection rectangle_selection=null;

    /** Liste des différents composants à dessiner sur la scène (le diagramme des classes) **/
    private LinkedList<IDessinableManipulable> figures = new LinkedList<IDessinableManipulable>();


    /**
     * Constructeur prenant en paramètre la palette de la scène
```

```
 * @param palette
 */
public SceneEditeur(ScenePalette palette) {
    this.palette = palette;
    // Permet de ne plus replacer le JTextArea
    this.setLayout(null);
}


[...]


/**
 * Affichage graphique de la scène
 * On surchage paintComponent de JPanel (la scène hérite de JPanel) afin de redéfinir un
 * affichage personalisé
 * @param Graphics contexte graphique
 */
public void paintComponent(Graphics g) {
    Graphics2D g2d=(Graphics2D)g;

    // Désactivation de l'anti-aliasing
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_OFF);
    g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
        RenderingHints.VALUE_TEXT_ANTIALIAS_OFF);
    // Demande de rendu rapide
    g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_SPEED);
    g2d.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
        RenderingHints.VALUE_COLOR_RENDER_SPEED);
    g2d.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
        RenderingHints.VALUE_FRACTIONALMETRICS_OFF);
    g2d.setRenderingHint(RenderingHints.KEY_DITHERING, RenderingHints.VALUE_DITHER_DISABLE);
    g2d.setColor(Color.white);
    g2d.fillRect(0,0,getWidth(),getHeight());

    // Affichage de tous les composants
    for(int i=figures.size()-1; i>=0; i--)
        figures.get(i).dessiner(g2d);

    // Affichage du cadre de sélection
    if(rectangle_selection!=null) rectangle_selection.dessiner(g2d);
}


/**
 * Permet de vérifier si une figure sort de la scène et redimensionne la scène
 * Repaint de la scène
 */
public void update() {
    int maxheight=0;
    int maxwidth=0;
    for(int i=figures.size()-1; i>=0;i--) {
        IDessinableManipulable ist = (IDessinableManipulable)figures.get(i);
        if(ist.getClass()==RepresentationNote.class
                || ist.getClass()==RepresentationClasse.class) {
            Dimension d = new Dimension(ist.getPosition().x+ist.getRectangle().width,
```

```
                ist.getPosition().y+ist.getRectangle().height);
            if(d.height>maxheight) maxheight=d.height;
            if(d.width>maxwidth) maxwidth=d.width;
        }
    }
    setPreferredSize(new Dimension(maxwidth,maxheight));
    setSize(maxwidth,maxheight);
    repaint();
}


public BufferedImage getImage() {
    int width = this.getWidth();
    int height = this.getHeight();

    BufferedImage tempImage= new BufferedImage(width+20, height+20, BufferedImage.TYPE_INT_RGB);
    Point pSource=new Point(0,0);
    Point pDestination=new Point(width, height);

    Graphics2D g = tempImage.createGraphics();
    g.setColor(Color.white);
    g.fillRect(0, 0, tempImage.getWidth(), tempImage.getHeight());

    this.paintAll(g);
    g.dispose();
    Point pHautGauche = new Point(width,width);
    Point pBasDroite = new Point(0, 0);;

    for (Iterator<IDessinableManipulable> iter = figures.iterator(); iter.hasNext();) {
        IDessinableManipulable figure = iter.next();
        if(figure.getClass()==RepresentationNote.class ||
                figure.getClass()==RepresentationClasse.class) {
            if(figure.getPosition().x<pHautGauche.x)
                pHautGauche.x=figure.getPosition().x;
            if(figure.getPosition().y<pHautGauche.y)
                pHautGauche.y=figure.getPosition().y;
            if(figure.getPosition().x+figure.getRectangle().width>pBasDroite.x)
                pBasDroite.x=figure.getPosition().x+figure.getRectangle().width;
            if(figure.getPosition().y+figure.getRectangle().height>pBasDroite.y)
                pBasDroite.y=figure.getPosition().y+figure.getRectangle().height;
        }
    }
    if(figures.size()>0) {
        pSource = pHautGauche;
        pDestination = pBasDroite;
    }

    BufferedImage imgfinalTemp = tempImage.getSubimage(pSource.x, pSource.y,
        pDestination.x-pSource.x, pDestination.y-pSource.y);
    BufferedImage img=new BufferedImage(imgfinalTemp.getWidth()+20, imgfinalTemp.getHeight()+20,
        BufferedImage.TYPE_INT_RGB);
    Graphics2D gti = img.createGraphics();
    gti.setColor(Color.white);
    gti.fillRect(0, 0, imgfinalTemp.getWidth()+20, imgfinalTemp.getHeight()+20);
    gti.drawImage(imgfinalTemp, null, 10, 10);
    return img;
```

```
}

[...]

/**
 * Permet de supprimer les composants selectionnes
 */
public void supprimerFiguresSelectionner() {
    // On supprime d'abord les liens sélectionnés pour éviter les problèmes de suppression en casc
    for (int i=0; i<figures.size(); i++) {
        IDessinableManipulable figure = figures.get(i);
        if(figure.getEstSelectionne()) {
            if(figure.getClass()==RepresentationHeritage.class) {
                ENV.FRAME.getControleur().supprimerHeritage((RepresentationHeritage)figure);
                i--;
            }
            if(figure.getClass()==RepresentationAgregation.class) {
                ENV.FRAME.getControleur().supprimerAggregation((RepresentationAgregation)figure);
                i--;
            }
        }
    }
    // On supprime les composants selectionnés
    for (int i=0; i<figures.size(); i++) {
        IDessinableManipulable figure = figures.get(i);
        if(figure.getEstSelectionner()==true) {
            if(figure.getClass()==RepresentationClasse.class) {
                ENV.FRAME.getControleur().supprimerClasseDuModel((RepresentationClasse)figure);
                i--;
            }
            if(figure.getClass()==RepresentationNote.class) {
                supprimerComposant(figure);
                i--;
            }
        }
    }
    repaint();
}

[...]

public void rajouterClasse(Point p) {
    ENV.FRAME.getControleur().ajouterClasse(p);
}

public void rajouterNote(Point p) {
    ENV.FRAME.getControleur().ajouterNote(p);
}

public void rajouterAggregation(Point p,RepresentationClasse fig_source,
        RepresentationClasse fig_dest) {
    ENV.FRAME.getControleur().ajouterAgregation(fig_source, fig_dest);
}
```

```
public void rajouterComposition(Point p,RepresentationClasse fig_source,
        RepresentationClasse fig_dest) {
    ENV.FRAME.getControleur().ajouterComposition(fig_source, fig_dest);
}

public void rajouterHeritage(Point p,RepresentationClasse fig_source,
        RepresentationClasse fig_dest) {
    ENV.FRAME.getControleur().ajouterHeritage(fig_source, fig_dest);
}

public void rajouterLienNote(RepresentationNote comp_source, Point point) {
    RepresentationLienNote rln=new RepresentationLienNote(comp_source, point);
    figures.add(rln);
}


/**
 * Permet de déplacer les composants sélectionnés suivant une touche directionnelle et une vitesse
 * @param toucheDirection direction du déplacement (Voir enumeration Touche)
 * @param nbrPixels nombre de pixels à avancer
 */
public void avancerComposant(Touche toucheDirection, int nbrPixels) {
    for (Iterator<IDessinableManipulable> iter = figures.iterator(); iter.hasNext();) {
        IDessinableManipulable composant = iter.next();
        if(composant.getEstSelectionner()) {
            Point p = new Point(composant.getPosition());
            if(toucheDirection==Touche.HAUT) p.y-=nbrPixels;
            if(toucheDirection==Touche.BAS) p.y+=nbrPixels;
            if(toucheDirection==Touche.DROITE) p.x+=nbrPixels;
            if(toucheDirection==Touche.GAUCHE) p.x-=nbrPixels;
            composant.setPosition(p);
        }
    }
    update();
}


/**
 * Permet d'aligner les composants suivant leurs bords
 */
public void alignerComposantsScene() {
    for (int j=0; j<figures.size(); j++) {
        IDessinableManipulable composant = figures.get(j);
        if(composant.getClass()==RepresentationClasse.class) {
            Point pRef= new Point(composant.getPosition());
            Dimension dimRef= new Dimension(composant.getDimension());

            for (int i=j+1; i<figures.size(); i++) {
                if(composant.getClass()==RepresentationClasse.class) {
                    IDessinableManipulable composantAutre = figures.get(i);
                    Point pComp= new Point(composantAutre.getPosition());
                    Dimension dimComp = new Dimension(composantAutre.getDimension());

                    if(((pComp.y>pRef.y) && (pComp.y<(pRef.y+dimRef.height)))||
                            (((pComp.y+dimComp.height)>pRef.y) &&
                            ((pComp.y+dimComp.height)<(pRef.y+dimRef.height)))) {
```

```
                            Point nouveauPoint = new Point(pComp.x, pRef.y);
                            // On vérifie si une figure ne possède pas dejà ses coordonnées
                            if(verifierCoordonnees(nouveauPoint))
                                composantAutre.setPosition(nouveauPoint);
                        }

                        if(((pComp.x>pRef.x) && (pComp.x<(pRef.x+dimRef.width)))||
                                (((pComp.x+dimComp.width)>pRef.x) &&
                                ((pComp.x+dimComp.width)<(pRef.x+dimRef.width)))) {
                            Point nouveauPoint = new Point(pRef.x, pComp.y);
                            // On verifie si une figure ne possède pas dejà ses coordonnées
                            if(verifierCoordonnees(nouveauPoint))
                                composantAutre.setPosition(nouveauPoint);
                        }
                    }
                }
            }
        }
        repaint();
    }


    /**
     * Permet de sélectionner tous les composants de la scène
     */
    public void selectionnerTousLesComposants() {
        for (int j=0; j<figures.size(); j++) {
            IDessinableManipulable composant = figures.get(j);
            composant.setEstSelectionne(true);
        }
        repaint();
    }


    [...]


}
```

B.3 L'exportation

B.3.1 Fichier XML

Voici le contenu du fichier XML permettant l'exportation dans le langage Java.

```
<?xml version="1.0" ?>

<langage nom="Java">
    <keywords>
        <class>class</class>
        <begin>{</begin>
        <end>}</end>
        <endl>;</endl>
        <public>public</public>
        <private>private</private>
        <protected>protected</protected>
```

```
        <heritage>extends</heritage>
        <interface>implements</interface>
        <abstract>abstract</abstract>
        <string>String</string>
        <boolean>boolean</boolean>
        <list>List</list>
    </keywords>
    <constructeurs>
        <implementationConstructeurPlusEvolue>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>Constructeur</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
            <it>tabulation</it><it>public</it><it>espace</it><it>nom</it><it>(</it>
            <attributs arborescence="unique"><premier>premierInitAttribut</premier>
            <autres>autresInitAttributs</autres></attributs><it>)</it><it>espace</it>
            <it>begin</it><it>newline</it>
            <classes arborescence="unique"><tous>initClasse</tous></classes>
            <attributs><tous>tousInitAttribut</tous></attributs>
            <it>tabulation</it><it>end</it><it>newline</it>
        </implementationConstructeurPlusEvolue>
    </constructeurs>
    <heritages>
        <declarationHeritage>
            <it>heritage</it>
            <it>espace</it>
            <it>nom</it>
        </declarationHeritage>
    </heritages>
    <agregations>
    </agregations>
    <attributs>
        <declarationAttribut>
            <it>tabulation</it><it>/**</it><it>espace</it><it>description</it><it>espace</it>
            <it>**/</it><it>newline</it>
            <it>tabulation</it><it>visibilite</it><it>espace</it><it>type</it><it>espace</it>
            <it>nom</it><it>endl</it><it>newline</it>
            <it>newline</it>
        </declarationAttribut>
        <getter>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>Retourne la valeur de</it>
            <it>espace</it><it>nom</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
            <it>tabulation</it><it>public</it><it>espace</it><it>type</it><it>espace</it>
            <it>get</it><capitalize>nom</capitalize><it>()</it><it>espace</it><it>begin</it>
            <it>newline</it>
            <it>tabulation</it><it>tabulation</it><it>return this.</it><it>nom</it>
            <it>endl</it><it>newline</it>
            <it>tabulation</it><it>end</it><it>newline</it>
            <it>newline</it>
        </getter>
        <setter>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>Permet d'affecter une valeur à</it>
            <it>espace</it><it>nom</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
```

```
            <it>tabulation</it><it>public</it><it>espace</it><it>void</it><it>espace</it>
            <it>set</it><capitalize>nom</capitalize><it>(</it><it>type</it><it>espace</it>
            <it>nom</it><it>)</it><it>espace</it><it>begin</it><it>newline</it>
            <it>tabulation</it><it>tabulation</it><it>this.</it><it>nom</it><it>espace</it>
            <it>=</it><it>espace</it><it>nom</it><it>endl</it><it>newline</it>
            <it>tabulation</it><it>end</it><it>newline</it>
            <it>newline</it>
        </setter>
        <premierInitAttribut>
            <it>type</it><it>espace</it><it>nom</it>
        </premierInitAttribut>
        <autresInitAttributs>
            <it>,</it><it>espace</it><it>type</it><it>espace</it><it>nom</it>
        </autresInitAttributs>
        <premierAttributSuper>
            <it>nom</it>
        </premierAttributSuper>
        <autresAttributsSuper>
            <it>,</it><it>espace</it><it>nom</it>
        </autresAttributsSuper>
        <tousInitAttribut>
            <it>tabulation</it><it>tabulation</it><it>this.</it><it>nom</it><it>espace</it>
            <it>=</it><it>espace</it><it>nom</it><it>endl</it><it>newline</it>
        </tousInitAttribut>
    </attributs>
    <parametres>
        <premierParametre>
            <it>type</it><it>espace</it><it>nom</it>
        </premierParametre>
        <autresParametres>
            <it>,</it><it>espace</it><it>type</it><it>espace</it><it>nom</it>
        </autresParametres>
    </parametres>
    <methodes>
        <implementationMethode>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>description</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
            <it>tabulation</it><it>visibilite</it><it>espace</it><it>typeRetour</it><it>espace</it>
            <it>nom</it><it>(</it><parametres><premier>premierParametre</premier>
            <autres>autresParametres</autres></parametres><it>)</it><it>espace</it><it>begin</it>
            <it>newline</it>
            <it>tabulation</it><it>tabulation</it><it>newline</it>
            <it>tabulation</it><it>end</it><it>newline</it>
            <it>newline</it>
        </implementationMethode>
        <implementationMethodeAbstraite>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>description</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
            <it>tabulation</it><it>abstract</it><it>espace</it><it>visibilite</it><it>espace</it>
            <it>typeRetour</it><it>espace</it><it>nom</it><it>(</it><parametres>
            <premier>premierParametre</premier><autres>autresParametres</autres></parametres>
            <it>)</it><it>endl</it><it>newline</it>
            <it>newline</it>
        </implementationMethodeAbstraite>
```

```
    </methodes>
    <classes>
        <initClasse>
            <it>tabulation</it><it>tabulation</it><it>super(</it><attributs arborescence="unique">
            <premier>premierAttributSuper</premier>
            <autres>autresAttributsSuper</autres></attributs><it>)</it><it>endl</it><it>newline</it>
        </initClasse>
    </classes>
    <fichiers>
        <classes nom="nomClasse" extension=".java">
            <it>/**</it><it>newline</it>
            <it>*</it><it>espace</it><it>description</it><it>newline</it>
            <it>*</it><it>espace</it><it>@author</it><it>espace</it><it>auteur</it><it>newline</it>
            <it>*/</it><it>newline</it>
            <it>abstract</it><it>espace</it><it>visibilite</it><it>espace</it><it>class</it>
            <it>espace</it><it>nom</it><it>espace</it><heritages arborescence="unique">
            <tous>declarationHeritage</tous></heritages><it>espace</it><it>begin</it><it>newline</it>
            <it>newline</it>
            <it>newline</it>
            <attributs visibilite="public"><tous>declarationAttribut</tous></attributs>
            <attributs visibilite="protected"><tous>declarationAttribut</tous></attributs>
            <attributs visibilite="private"><tous>declarationAttribut</tous></attributs>
            <it>newline</it>
            <constructeurs>implementationConstructeurPlusEvolue</constructeurs><it>newline</it>
            <methodes abstraite="oui">
                <tous>implementationMethodeAbstraite</tous>
            </methodes>
            <methodes visibilite="public" abstraite="non">
                <tous>implementationMethode</tous>
            </methodes>
            <methodes visibilite="protected" abstraite="non">
                <tous>implementationMethode</tous>
            </methodes>
            <methodes visibilite="private" abstraite="non">
                <tous>implementationMethode</tous>
            </methodes>
            <attributs><tous>getter</tous></attributs><attributs><tous>setter</tous></attributs>
            <it>newline</it>
            <it>end</it><it>newline</it>
        </classes>
        <autre nom="Main" extension=".java">
            <it>public</it><it>espace</it><it>class</it><it>espace</it><it>Main</it><it>espace</it>
            <it>begin</it><it>newline</it>
            <it>newline</it>
            <it>tabulation</it><it>/**</it><it>newline</it>
            <it>tabulation</it><it>*</it><it>espace</it><it>Main</it><it>newline</it>
            <it>tabulation</it><it>*/</it><it>newline</it>
            <it>tabulation</it><it>public static void main(String[] args)</it><it>espace</it>
            <it>begin</it><it>newline</it>
            <it>tabulation</it><it>tabulation</it><it>newline</it>
            <it>tabulation</it><it>end</it><it>newline</it>
            <it>newline</it>
            <it>end</it>
        </autre>
    </fichiers>
</langage>
```

B.3.2 Classe GenerationCode

Voici la classe de qui permet la génération. La seule fonction statique laissée dans cet extrait est celle permettant la génération d'une méthode en suivant une structure XML passée en paramètre.

```
package Exportation;

import [...]


/**
 * Permet de générer les différents fichiers selon un fichier XML ayant une structure particulière
 * (comprend des méthodes statiques permettant de générer les objets UML)
 * @author Fabien HERVOUET <fabien.her@gmail.com>
 */
public class GenerationCode {

    /** Mots-clefs pour l'exportation du langage **/
    protected Keywords kw;

    /** Structure XML **/
    protected Element noeudPere;

    /** Noeud des structeurs XML relatives aux différents objets **/
    private static Element clsStruct;
    private static Element attStruct;
    private static Element methStruct;
    private static Element paramStruct;
    private static Element herStruct;
    private static Element constStruct;
    private static Element agregStruct;

    /** Diagramme **/
    private static DiagrammeClasses diagramme;


    /**
     * Constructeur par fichier XML
     */
    private GenerationCode(String fichier, DiagrammeClasses d) {
        Document document = new Document();
        SAXBuilder sxb = new SAXBuilder();
        // on initialise le document avec le chemin du fichier
        try {
            document = sxb.build(new File(fichier));
        } catch(Exception e) {
            //ENV.LOG.afficherErreur("Erreur de lecture du fichier XML : " + fichier);
        }
        // on initialise la racine par rapport au document parsé
        this.noeudPere = document.getRootElement();
        // on initialise les mots-clefs
        this.kw = new Keywords(this.noeudPere.getChild("keywords"));
        // affectation des structures
        clsStruct = this.noeudPere.getChild("classes");
        paramStruct = this.noeudPere.getChild("parametres");
        attStruct = this.noeudPere.getChild("attributs");
        methStruct = this.noeudPere.getChild("methodes");
```

```
    herStruct = this.noeudPere.getChild("heritages");
    constStruct = this.noeudPere.getChild("constructeurs");
    agregStruct = this.noeudPere.getChild("agregations");
    diagramme = d;
}


[...]


/**
 * Permet de récupérer la déclaration de la méthode
 * @param nomClasse Nom de la classe à laquelle la méthode appartient
 * @param m Méthode dont il faut générer le code
 * @param gCode
 * @param methStruct Noeud XML qui traite de la génération d'une méthode
 */
private static String genererMethode(String nomClasse, Methode m, GenerationCode gCode,
        Element methStruct) {
    String codeFinal = "", code = "";
    Element premier, autres, tous;
    // booléen permettant de pas mettre des espaces en trop (si un élément est vide, et que
    // l'identifiant d'après est un espace, on n'en tient pas compte
    boolean vide = false;
    for (Object it : methStruct.getChildren()) {
        code = "";
        Element e = (Element)it;
        // on récupère le nom et le texte de la balise
        String name = e.getName();
        String str = e.getTextNormalize();
        premier = null; autres = null; tous = null;
        // on essaye de récupérer tout ce qui est possible
        premier = e.getChild("premier");
        autres = e.getChild("autres");
        tous = e.getChild("tous");
        // si on demande la liste des paramètres
        if (name.equals("parametres")) {
            if (!m.getParametres().isEmpty()) {
                // si il y a une différenciation (syntaxique) entre le premier et les
                // autres attributs
                if (premier != null && autres != null) {
                    // on traite le premier paramètre
                    code += genererVariable(nomClasse, m.getParametres().get(0),
                        gCode, paramStruct.getChild(premier.getValue()));
                    // puis les autres
                    for (int i = 1; i < m.getParametres().size(); i++) {
                        code += genererVariable(nomClasse, m.getParametres().get(i),
                            gCode, paramStruct.getChild(autres.getValue()));
                    }
                }
                // sinon on les affiche tous en suivant le modèle donné
                else if (tous != null) {
                    for (Variable v : m.getParametres()) {
                        code += genererVariable(nomClasse, v, gCode,
                            paramStruct.getChild(tous.getValue()));
                    }
```

```
        }
        vide = false;
    }
    else {
        vide = true;
    }
}
// sinon si on demande le type retourné par la méthode
else if (str.equals("typeRetour")) {
    // si le type n'est pas vide
    if (!m.getTypeRetour().isEmpty()) {
        // on regarde si le type demande à être "casté" dans les mots-clefs
        if (gCode.getKw().getKeyword(m.getTypeRetour()) != null) {
            code += gCode.getKw().getKeyword(m.getTypeRetour());
        }
        else {
            code += m.getTypeRetour();
        }
        vide = false;
    }
    else {
        vide = true;
    }
}
// sinon si on demande le nom de la méthode
else if (str.equals("nom")) {
    if (!m.getNom().isEmpty()) {
        code += m.getNom();
        vide = false;
    }
    else {
        vide = true;
    }
}
// sinon si on demande la visibilité de la méthode
else if (str.equals("visibilite")) {
    if (!m.getVisibilite().isEmpty()) {
        code += m.getVisibilite();
        vide = false;
    }
    else {
        vide = true;
    }
}
// sinon si on demande la description de la méthode
else if (str.equals("description")) {
    if (!m.getDescription().isEmpty()) {
        code += m.getDescription();
        vide = false;
    }
    else {
        vide = true;
    }
}
// sinon si on demande le nom de la classe à laquelle la méthode appartient
else if (str.equals("nomClasse")) {
```

```
                code += nomClasse;
                vide = false;
            }
            // sinon si on demande un retour à la ligne
            else if (str.equals("newline")) {
                code += "\n";
                vide = false;
            }
            // sinon si on demande une tabulation, on lance la méthode qui vérifie si
            // l'utilisateur veut des espaces ou des tabulations
            else if (str.equals("tabulation")) {
                code += tabulation(gCode.getKw().getKeyword("tabulation"));
                vide = false;
            }
            // sinon si on demande un espace
            else if (str.equals("espace")) {
                if (!vide) {
                    code += " ";
                }
                else {
                    vide = false;
                }
            }
            // sinon on regarde si c'est un mot-clef
            else if (gCode.getKw().getKeyword(str) != null) {
                code += gCode.getKw().getKeyword(str);
                vide = false;
            }
            else {
                code += str;
                vide = false;
            }
            // on formatte le code (prise en compte du upper, lower, etc...)
            codeFinal += formaterCode(code, name);
        }
        return codeFinal;
    }

    [...]


}
```

B.4 L'importation

Le fichier qui suit le *.lex* destiné à la reconnaissance et au traitement du contenu d'un script Python.

```
/**
  * Fichier Python.flex permettant de récupérer tous les éléments
  * des classes contenues dans les fichiers Python à scanner
  * @author Laurent Muthelet
  */

// Partie "code utilisateur"
```

```
package ReverseEng;

import [...]

%%
// Partie "Options et Déclarations"
%public
%class ReverseEngPython
%standalone
%unicode

%{
/* Déclaration de variables globales utilisées tout au long du fichier */
boolean classeFerm = true;
ArrayList<Classe> classeImbriquee = new ArrayList<Classe>();
ArrayList<String> classeTab = new ArrayList<String>();
Classe c;
Methode m;
Variable v;
String visibilite, type, nom;
String description = "";
ArrayList<Attribut> liste_attributs = new ArrayList<Attribut>();
ArrayList<Methode> liste_methodes = new ArrayList<Methode>();
ArrayList<Classe> liste_classes = new ArrayList<Classe>();
char loc;
int i, cpt, cpt2;
DiagrammeClasses d = new DiagrammeClasses();
ArrayList<CoupleStringString> heritage = new ArrayList<CoupleStringString>();
static ArrayList<CoupleStringString> heritageStat;
int nbreTab = 0;
int nbreTabclass = 0;

/**
* Permet de renvoyer un diagramme à partir du fichier scanné
* @param classes Chemin du fichier à scanner
*/
public static DiagrammeClasses getDiagramme(String classes) {
    DiagrammeClasses DCloc = new DiagrammeClasses();
    ReverseEngPython scanner = null;
    String nom_classe = "";
    if (classes.toLowerCase().contains("/")) {
        nom_classe = classes.substring(classes.lastIndexOf('/') + 1);
    }
    else {
        nom_classe = classes.substring(classes.lastIndexOf('\\') + 1);
    }

    // On scanne le fichier
    try {
        scanner = new ReverseEngPython( new java.io.FileReader(classes) );
        while ( !scanner.zzAtEOF ) scanner.yylex();
        DCloc.setClasses(scanner.getDiagrammeClasse());
        ENV.LOG.afficherLigne("La classe \"" + nom_classe + "\" a été scannée avec succès.");
        heritageStat = scanner.getHeritage();
    }
    catch (java.io.FileNotFoundException e) {
```

```
        ENV.LOG.afficherErreur("ATTENTION : Le fichier contenant la classe \"" + nom_classe
            + "\" est introuvable !");
    }
    catch (java.io.IOException e) {
        ENV.LOG.afficherErreur("ATTENTION : Le scan de la classe \"" + nom_classe
            + "\" a échoué !");
    }
    catch (Exception e) {
        ENV.LOG.afficherErreur("ATTENTION : Le scan de la classe \"" + nom_classe
            + "\" a échoué !");
    }
    return DCloc;
}


/**
* Retourne les classes contenues dans le fichier à scanner
* @return ArrayList<Classe>
*/
public ArrayList<Classe> getDiagrammeClasse() {
    return d.getClasses();
}


/**
* Retourne les héritages contenus dans le fichier à scanner
* @return ArrayList<CoupleStringString>
*/
public ArrayList<CoupleStringString> getHeritage() {
    return heritage;
}


/**
* Retourne les héritages (statiques) contenus dans le fichier à scanner
* @return ArrayList<CoupleStringString>
*/
public static ArrayList<CoupleStringString> getHeritageStat() {
    return heritageStat;
}


/**
* Fonction permettant de recuperer les elements (visibilite,
* type, nom) d'un attribut ou d'une methode
* et de les inserer aux listes d'attributs et de methodes
* @param i Entier désignant l'index de fin de la ligne à scanner
* @param element Indique quel type d'élément on traite (attribut ou méthode)
*/
public void ReconnaitVar(int i) {
    i--;

    // On récupère le nom de la méthode et on l'insère à la liste des méthodes
    loc = yytext().charAt(i);
    type = "";
    while (loc == ' ' || loc == '\t' || loc == '\n' || loc == '\r' || loc == '\f')
```

```
    {
        i--;
        loc = yytext().charAt(i);
    }

    cpt = i+1;
    while (i != 0 && loc != ' ' && loc != '\t' && loc != '\n' && loc != '\r' && loc != '\f')
    {
        i--;
        loc = yytext().charAt(i);
    }
    if (i != 0) i++;
    nom = yytext().substring(i, cpt);
    nom = nom.replace(" ","");
    nom = nom.replace("\t","");
    visibilite = "public";
    m = new Methode();
    m.setVisibilite(visibilite);
    m.setTypeRetour(type);
    m.setNom(nom);
    m.setAbstraite(false);
    m.setDescription(description);
    liste_methodes.add(m);
    description = "";
}
%}

// Déclaration des types d'expression à scanner
FinDeLigne = \r|\n|\r\n
Commentaires = "#" .* | "'''" [^'] ~"'''"
Chaine = \" ([^\n\"] | \\\" )* \"
Blancs = {FinDeLigne} | [ \t\f]
BlancsOuRien = {Blancs}|""
TabOuRien = [ \t]|""
Ident = [:jletter:][:jletterdigit:]*

%%
// Partie "Règles Lexicales"

// Reconnaissance de commentaires (peut-être la description d'éléments comme les
// attributs ou les méthode)
{Commentaires} {
    /**
    * On récupère les commentaires liés à la classe ou a une methode
    */
    loc = yytext().charAt(0);
    if (loc == '#')
    {
        description = yytext().substring(1);
    }
    else
    {
        description = yytext().substring(3,yytext().length()-3);
    }
    if (!description.equals(""))
    {
```

```
        description = description.replace("\r"," ");
        description = description.replace("\n"," ");
        description = description.replace("  "," ");
    }
}

// Reconnaissance de blancs
{Blancs} {} // on ignore

// Reconnaissance de chaînes de caractères
{Chaine} {} // on ignore

// Reconnaissance d'une classe
{TabOuRien} + "class" + {Blancs} + {Ident} + (("(" + [^)]* + ")")|({BlancsOuRien} + ":")) {
    /**
    * On récupère les éléments (nom, description) de la classe
    */
    nbreTab = 0;
    i = 0;
    loc = yytext().charAt(i);
    while (loc == ' ' || loc == '\t')
    {
        if (loc == '\t') nbreTab += 8;
        else nbreTab++;
        i++;
        loc = yytext().charAt(i);
    }
    if ((nbreTab > nbreTabclass) && !classeFerm)
    {
        classeImbriquee.add(c);
        classeTab.add(String.valueOf(nbreTabclass));
    }
    classeFerm = false;
    if (!liste_attributs.isEmpty() || !liste_methodes.isEmpty())
    {
        c.setMethodes(liste_methodes);
        c.setAttributs(liste_attributs);
        liste_classes.add(c);
        liste_methodes = new ArrayList<Methode>();
        liste_attributs = new ArrayList<Attribut>();
    }
    cpt = 0;

    // On récupère le nom de la classe et on ajoute la classe à la liste de classes
    i = yytext().indexOf("class") + 6;
    loc = yytext().charAt(i);
    while (loc == ' ' || loc == '\t' || loc == '\n' || loc == '\r' || loc == '\f') {
        i++;
        loc = yytext().charAt(i);
    }
    nom = yytext().substring(i);
    while (loc != ' ' && loc != '\t' && loc != '\n' && loc != '\r' && loc != '\f' && loc
        != '(' && loc != ':') {
        cpt++;
        i++;
        loc = yytext().charAt(i);
```

```
    }
    nom = nom.substring(0,cpt);
    nom = nom.replace(" ","");
    nom = nom.replace("\t","");
    c = new Classe(description,nom,"public");
    c.setAbstraite(false);
    d.ajouterClasse(c);

    // Gestion des classes Mère et Fille en cas d'héritage
    if (yytext().toLowerCase().contains("(")) {
        boolean vide = true;
        i = yytext().indexOf("(") + 1;
        loc = yytext().charAt(i);
        while (loc != ')') {
            if (loc != ' ' && loc != '\t' && loc != '\n' && loc != '\r' && loc != '\f') {
                vide = false;
            }
            i++;
            loc = yytext().charAt(i);
        }
        i = yytext().indexOf("(") + 1;
        loc = yytext().charAt(i);
        cpt = i;
        String nomClasseMere;
        while (loc != ')') {
            if (loc == ',') {
                nomClasseMere = yytext().substring(cpt,i);
                nomClasseMere = nomClasseMere.replace(" ","");
                heritage.add(new CoupleStringString(nom, nomClasseMere));
                cpt = i + 1;
            }
            i++;
            loc = yytext().charAt(i);
        }
        if (!vide) {
            nomClasseMere = yytext().substring(cpt,i);
            nomClasseMere = nomClasseMere.replace(" ","");
            heritage.add(new CoupleStringString(nom, nomClasseMere));
        }
    }
    description = "";
    nbreTabclass = nbreTab;
}

// Reconnaisance des constructeurs contenant en paramètres les attributs de la classe
{TabOuRien} + "def" + {Blancs} + "__init__" + {BlancsOuRien} + "(" + [^)]* + ")" {
    /**
    * On récupère le nom des attributs passés en paramètres dans le constructeur
    * car ce sont les attributs de la classe
    */
    int nbreTabloc = 0;
    i = 0;
    loc = yytext().charAt(i);
    while (loc == ' ' || loc == '\t') {
        if (loc == '\t') nbreTabloc += 8;
        else nbreTabloc++;
```

```
    i++;
    loc = yytext().charAt(i);
}
if (nbreTabloc > nbreTabclass) {
    if (!yytext().toLowerCase().contains("#") && !yytext().toLowerCase().contains("’’’")
        && !yytext().toLowerCase().contains("\"\"")) {
        i = yytext().indexOf("(") + 1;
        cpt = i;
        int cpt2 = i;
        type = "";
        visibilite = "private";
        while (loc != ’)’) {
            if (loc == ’,’) {
                nom = yytext().substring(cpt, i);
                if (!nom.equals("self")) {
                    v = new Variable(type, nom);
                    Attribut a = new Attribut(visibilite, type, nom , "");
                    liste_attributs.add(a);
                }
                cpt = i+1;
                cpt2 = i+1;
            }
            i++;
            loc = yytext().charAt(i);
        }
        nom = yytext().substring(cpt2, i);
        nom = nom.replace(" ","");
        nom = nom.replace("\t","");
        v = new Variable(type, nom);
        if (!nom.equals("self")) {
            Attribut a = new Attribut(visibilite, type, nom , description);
            liste_attributs.add(a);
        }
    }
}
else if (!classeImbriquee.isEmpty() && nbreTabloc != 0) {
    c.setMethodes(liste_methodes);
    c.setAttributs(liste_attributs);
    if (!liste_classes.contains(c)) liste_classes.add(c);
    c = classeImbriquee.get(classeImbriquee.size()-1);
    liste_methodes = c.getMethodes();
    liste_attributs = c.getAttributs();
    classeImbriquee.remove(c);
    nbreTabclass = Integer.parseInt(classeTab.get(classeTab.size()-1));
    classeTab.remove(classeTab.get(classeTab.size()-1));
    i = yytext().indexOf("(") + 1;
    cpt = i;
    int cpt2 = i;
    type = "";
    visibilite = "private";
    while (loc != ’)’) {
        if (loc == ’,’) {
            nom = yytext().substring(cpt, i);
            if (!nom.equals("self")) {
                v = new Variable(type, nom);
                Attribut a = new Attribut(visibilite, type, nom , "");
```

```
                    liste_attributs.add(a);
                }
                cpt = i+1;
                cpt2 = i+1;
            }
            i++;
            loc = yytext().charAt(i);
        }
        nom = yytext().substring(cpt2, i);
        nom = nom.replace(" ","");
        nom = nom.replace("\t","");
        v = new Variable(type, nom);
        if (!nom.equals("self")) {
            Attribut a = new Attribut(visibilite, type, nom , description);
            liste_attributs.add(a);
        }
    }
    m = new Methode();
}

// Reconnaissance des méthodes de la classe
{TabOuRien} + "def" + {Blancs} + {Ident} + {BlancsOuRien} + "(" {
    /**
    * Appel de la fonction ReconnaitVar() pour inserer les elements
    * (nom, description) d'une methode
    */
    int nbreTabloc = 0;
    i = 0;
    loc = yytext().charAt(i);
    while (loc == ' ' || loc == '\t') {
        if (loc == '\t') nbreTabloc += 8;
        else nbreTabloc++;
        i++;
        loc = yytext().charAt(i);
    }
    if (nbreTabloc > nbreTabclass) {
        if (!yytext().toLowerCase().contains("#") && !yytext().toLowerCase().contains("'''") && !yytex
            i = yytext().indexOf("(");
            ReconnaitVar(i);
        }
    }
    else if (!classeImbriquee.isEmpty() && nbreTabloc != 0) {
        c.setMethodes(liste_methodes);
        c.setAttributs(liste_attributs);
        if (!liste_classes.contains(c)) liste_classes.add(c);
        c = classeImbriquee.get(classeImbriquee.size()-1);
        liste_methodes = c.getMethodes();
        liste_attributs = c.getAttributs();
        classeImbriquee.remove(c);
        nbreTabclass = Integer.parseInt(classeTab.get(classeTab.size()-1));
        classeTab.remove(classeTab.get(classeTab.size()-1));
        i = yytext().indexOf("(");
        ReconnaitVar(i);
    }
}
```

```
// Reconnaissance des variables passées en paramètres des méthodes de la classe
{Ident} + {BlancsOuRien} + (","|")") {
    /**
    * Récupération du nom des paramètres des méthodes
    */
    cpt = 0;
    i = 0;
    cpt = 0;
    type = "";

    // On récupère le nom de la variable passée en paramètre
    nom = yytext().substring(i);
    loc = yytext().charAt(i);
    while (loc != ',' && loc != ')') {
        cpt++;
        i++;
        loc = yytext().charAt(i);
    }
    nom = nom.substring(0,cpt);
    nom = nom.replace(" ","");
    nom = nom.replace("\t","");
    if (!nom.equals("self")) {
        v = new Variable(type,nom);
        m.ajouterParametre(v);
    }
    if (loc == ')') m = new Methode();
}

// Fin du fichier atteinte, on ajoute la dernière classe trouvée
<<EOF>> {
    c.setMethodes(liste_methodes);
    c.setAttributs(liste_attributs);
    liste_classes.add(c);
    return 0;
}
```